UNDERSTANDING THE STANTEC-ZEBRA

by

CHENGETAI DANSEL KADENGE

846900

SUPERVISOR: PROFESSOR JOHN TUCKER

SEPTEMBER 2016

This project dissertation is submitted to the Swansea University in partial fulfilment for the Degree of Master of Science in Advance Software Technology

Department of Computer Science Swansea University.

DECLARATIONS AND STATEMENTS

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

This dissertation is the result of my own independent work / investigation, except where otherwise stated. Other sources are specifically acknowledged by clear cross referencing to author, work and page(s) using the bibliography / references section. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this dissertation and the degree examination as a whole.

I hereby give consent for my dissertation, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Name: CHENGETAI DANSEL KADENGE Signature: Kadenge

Date: 29/09/2016

This project dissertation is submitted to the Swansea University in partial fulfilment for the Degree of Master of Science in Advance Software Technology

ACRONYMS AND ABBREVATIONS

EDSAC	Electronic Delay Storage Automatic Calculator
ENIAC	Electronic Numerical Integrator and Calculator
SSEM	Small Scale Experimental Machine
STANTEC	Standard Telephones and Cables Limited
UCD	User-centred design
ZEBRA	Zeer Eenvoudige Binaire Reken-Automat

TABLE OF CONTENTS

List of Figures

List of Tables

ACKNOWLEDGMENTS

Even though the amount of effort put into this dissertation was enormous on a personal level, it has been influenced in many ways, all of them good, by many individuals. Because the roles that each of these individuals played were very significant, I would like to take the time to thank as many of them as I can remember here.

First and foremost, I would like to thank God for giving me the opportunity to study this course and guide me through the journey of being a Masters student at Swansea University. Without his guidance I would not have the strength to endure and conquer my trials.

I would like to give a big thank you to my supervisor, Professor John Tucker. Without his enthusiasm in early computer technology I would not have been able to do anything on this project. He reminded me of technologies that I never knew existed and I am proud and happy to know about them now. His constant monitoring of my progress and tasks that he gave me to do always kept me on my toes. I want to thank him so much for the belief that he had in me even when it looked so impossible to do the project. Also many thanks go to Mrs. Jill Edwards for assisting me to communicate efficiently and effectively with my supervisor. It would not have been possible without her.

I would also like to take this time to thank my friends Penelope, Nyasha, Tafadzwa, Natalie, Takunda, Denzel, Dimitris, Lin, Robert, Beveline, Kudzanai Kapurura, Nab, Rumbidzai, Wilfred and Emily Gusta for the support they gave me during the project development process. Without their push and helping hand I would not be at the level at I am right now.

I would also like to thank Mrs. Chiedza Makoni and Munyaradzi Kadenge for their continued support and always checking up on me if I was doing school work or not.

Finally, a big thank you goes to my parents, Levee Kadenge and Asiusinaye Kadenge for their love and care and trust in me to send me to the UK to study a Master's Program in Computer Science. In my wildest dreams I had not envisioned myself as studying in the UK but with their unending love it materialised. I would also like to give myself a pat on the back for not giving up for I had ventured into uncharted territories but I conquered in the end.

ABSTRACT

This dissertation focuses on the Stantec-ZEBRA computer from the 1950s designed by Willian van de Poel. This computer is a mystery to contemporary computer scientists but this is a computer that was used to make many important calculations, such as calculating the trajectory with which an aircraft has to take in order for it to land safely on the runway. At that time, calculations would take hours of number crunching by this computer but with today's computers and resources it can be done within the blink of an eye.

The goal is to try and discover how this ZEBRA worked, how it performed its calculations, and what type of programming language was used to write programs that ran on this computer. This is achieved by creating new programs written in a language that was compatible to the computer. These programs will be created for the ZEBRA emulator created by Don Hunter using the Simple Code programming language.

This document is divided into five parts. The first part focuses on the history of early computers focusing mainly on the Stantec-ZEBRA and emulators in general. The second part is based on the authors interview with Rod Delamere, one of the people to use the actual ZEBRA computer in the 1950s. The third part dives deeper into the details of the computer giving a literature review and describing emulators and the emulation process. The fourth part illustrates the project development process, giving the methodology that is used to prepare this thesis and specifying the technology used to develop the programs to run on the emulator. The fifth and final chapter gives the outcome of the whole process and stating the challenges that were met and how they were overcome and giving a final conclusion to the thesis itself.

College of Science
Coleg Gwyddoniaeth
Swansea University
Prifysgol Abertawe

Part I

Overview

➢ Chapter 1: INTRODUCTION

1 INTRODUCTION

It is good practise to preserve and maintain records of technological advancements as they are a symbol of achievement and breakthrough. As new, faster and cheaper software and hardware is created society tends to forget the origins of these technologies, which are rendered obsolete. Digital preservation comes into play by trying to conserve these achievements by documenting them. Digital preservation in itself is a formal endeavour to ensure that digital information of continuing value remains accessible and usable (User, 2016).

(The encyclopaedia of computer languages, no date) In 1976, at the History of Computing Conference in Los Alamos, Richard Hamming described why we might be interested in the history of computing: "we would know what they thought when they did it". We need to know why the people who designed computers thought the way they did. When they designed them, they made conscious choices, which we have to live with. And they made those choices for a wide variety of reasons, as many as the reasons for which they felt the need to create a computer in the first place. By knowing this, we can see the path in which those people had envisioned for the future and what expectations they had. As the world becomes more dependent on software and computers we find that the core of that software and computer still remain a mystery to even their users, the programmers. By paying attention to the origin, rise and fall of each of the computers, we may learn why the inventors made their decisions.

Many computers and programs have been created since the start of 1940 and the Electronic Numerical Integrator and Calculator (ENIAC) is commonly called the first computer because it was the first fully functional electronic computer. Computers during this time would fill an entire room needing ventilation and many vacuum and mercury tubes to run and store programs in. As big as they were, they would probably compute simple calculations or to output data on paper tape which will need further interpretation in the end. Apart from having these computers, this did not stop other people from designing and coming up with smaller and more powerful computers of their own.

This thesis will focus on one of the early computers known as the Stantec-ZEBRA. This computer is one of the many that were designed during the period of 1952-1958, including the EDSAC and Manchester "Baby" commonly known as Small Scale Experimental Machine. The ZEBRA is among the first commercial computers to be developed for use in Universities, Technical Colleges and Laboratories. Before the Stantec-ZEBRA, computers were only used by scientists and access was restricted. Willem van der Poel made the design of the ZEBRA as his PhD thesis in 1956 at the University of Amsterdam. ZEBRA is an acronym for the Dutch words ***Zeer Eenvoudige Binaire Reken-Automat***, which means Very Easy Binary Calculating Machine (Society, no date).

Around 58 of van der Poel's computers were built at the Standard Telephones and Cables company in Newport and most of them were shipped out to other countries and a small number was left in Wales the area of its birth. One of the remaining latest version of the ZEBRA computer is housed at the National Museum of Wales, and the aim of this project is to investigate the Stantec-ZEBRA emulator and bring back its performances and activities it used to carry out during its time.

1.1 Emulators

The Stantec-ZEBRA is among the first electronic computer to be used outside the scientific area and introduced into the academic and economic areas. The latest ZEBRA is held at The National Museum of Wales and it is not in functional condition. Other early computers such as the Small Scale Experimental Machine (SSEM) also known as the Manchester "Baby" and the Electronic Delay Storage Automatic Calculator (EDSAC) have been revived and maintained electronically through the use of emulators. These emulators run the programs that were previously run on the early computers. Having an emulator allows the current generation to have a feel of how the pioneering computers worked since modern computers have abstracted a lot of information. See figures 1 and 2 below showing the Manchester "Baby" and the EDSAC emulators respectively.

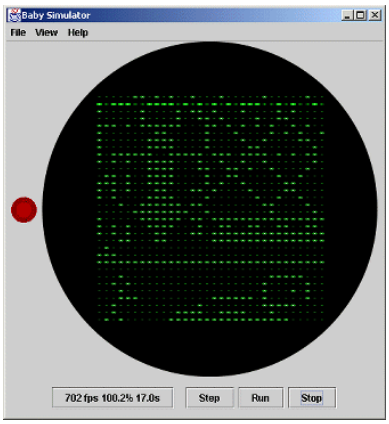


Figure 1.1: *Manchester "Baby" emulator*. Image source http://www.davidsharp.com/baby/

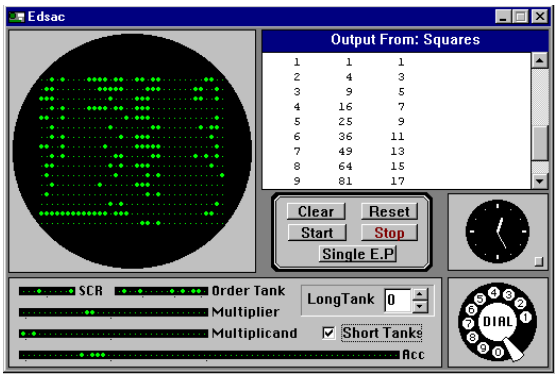


Figure 1.2: *EDSAC emulator*. Image source http://www.computerconservationsociety.org/software/edsac/base.htm

In addition, projects are underway to revive the hardware components of these early machines and one example is of the EDSAC Project at the National Museum of Computing in United Kingdom. Unfortunately, the Stantec-ZEBRA computer is in a condition that needs repair and the focus of this project is to help motivate the revival of this computer through giving a detailed understanding of the functionality of this early computer. Fortunately, enough, there exists an emulator for the ZEBRA and is available for download on the Computer Conservation Society website. Emulation refers to the ability of a computer program in an electronic device to imitate another program or device.

Some other programs of similar nature are BlueStakes, Andy and DOSBox. These are examples of the many, successful emulations that have happened up to this day. Blue Stakes and Andy are programs that emulate the Android mobile device so you can run applications such as WhatsApp and Candy crush on your computer as if it were a mobile or tablet.

DOSBox emulates the command-line interface of DOS and you can enter commands and run programs from there.

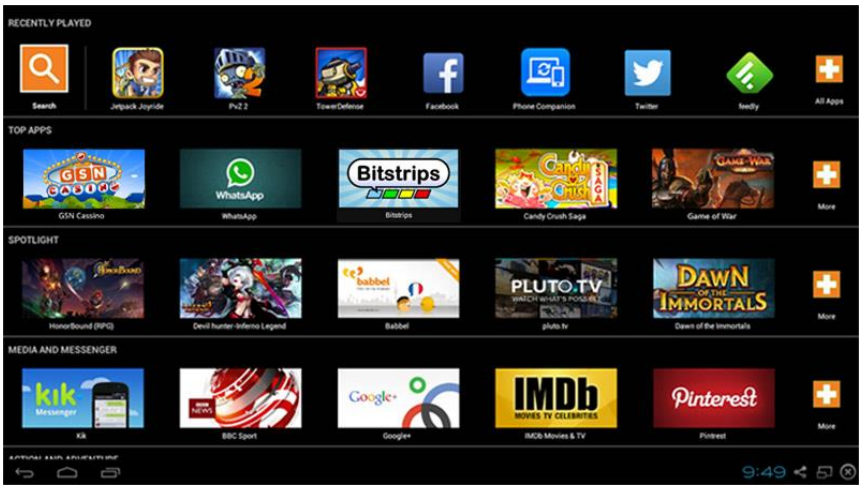


Figure 1.3 *BlueStakes*. Image source http://www.download-bluestacks.com/

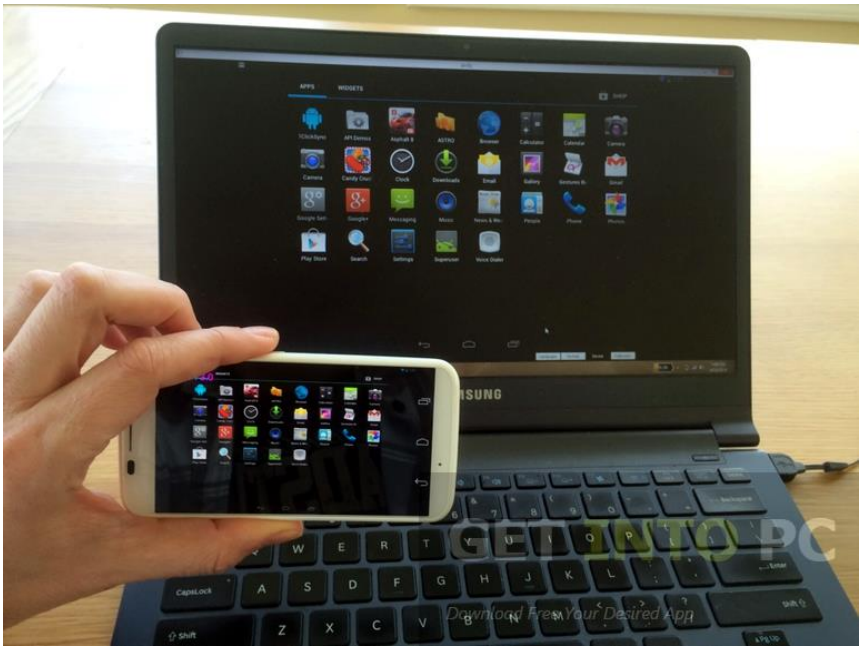


Figure 1.4 *Andy*. Image source http://getintopc.com/softwares/emulators/andy-android-emulator-free-download/

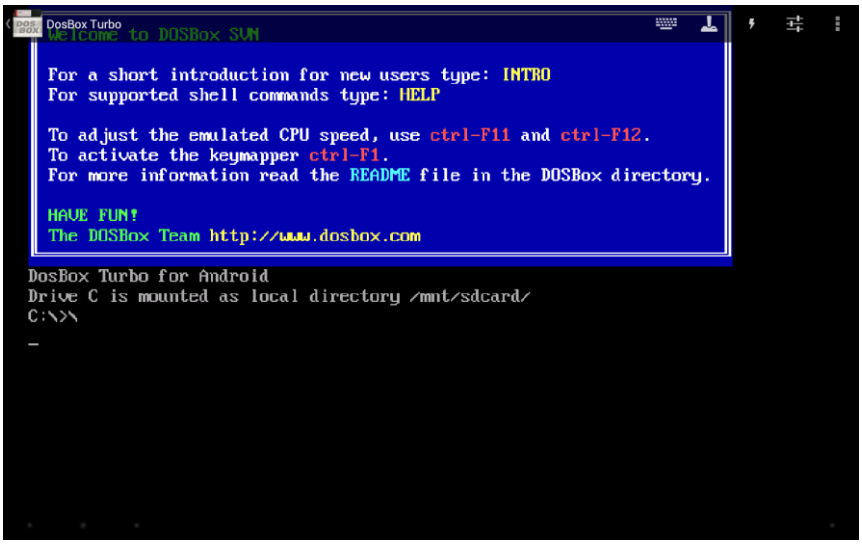


Figure 1.5 *DOSBox*

1.2 Aims

The primary focus of this research project is:

1. To find and bring together information that enables us to reconstruct the operation of an early computer, the Stantec-ZEBRA.
2. To raise awareness of the value of reviving old machines and legacy machines such as the Stantec-ZEBRA and to develop an appropriate methodology for such conservation.

1.3 Objectives

1. To master software engineering tools and methods of programming in Simple Code Language.
2. To write custom code in Simple Code Language that runs on the emulator for the ZEBRA computer.
3. To understand fine details of computer architecture and memories and machine code programming.
4. Be able to interpret some sample Simple Code programs written by Don Hunter

1.4 Methodology for the Stantec-ZEBRA

A work breakdown structure is the decomposition of the total scope of work to be carried out in order to accomplish the project objectives and create the required deliverables. By splitting up the work, a clear path for accomplishing the objectives is laid out in a clear manner. Below is the WBS that will be followed throughout the project development process.

The work structure of this project intends to start with learning the commands that were run on the Stantec-ZEBRA machine. How those commands executed the input that the computer was given and to study the declarations and definitions that are found in the commands. After learning the commands, the second step will be to find the language that is supported by those commands. Languages such as Simple Code and Normal Code were in existence during the 1950s and the target is to try and write a program in Simple code that will be able to run on an emulator of the ZEBRA that was designed by Don Hunter.

The best explanation of Normal Code is given in an article called Stantec-ZEBRA and it mentioned that, the structure of the Normal (machine) Code is based on a novel idea. Single letters specify basic operations such as add, test, store; but there are 15 such letters (called function digits) and these may be used in any combination so that the programmer may construct thousands of different instructions. It is possible to instruct the machine to add, transfer, shift, modify and test "all at the same time", thus making the effective speed of operation of the computer greater than the intrinsic electronic speed would suggest (Part17, no date).

At the same time of learning the programming language to be used to create the emulator and its commands, designing and writing of sample programs will also take place. By looking at other similar emulators like the Manchester baby and EDSAC, design structure of the ZEBRA can be deduced. After learning and design, implementation can then take place and the work put in will be on accomplishing a functional program written in Simple Code that runs on the ZEBRA emulator.

On the other side, working concurrently with code learning and program writing, focus will be also put on writing the project documentation and possibly having a tour to see the

hardware of the existing ZEBRA machine. After that the priority will be to raise awareness on the importance of

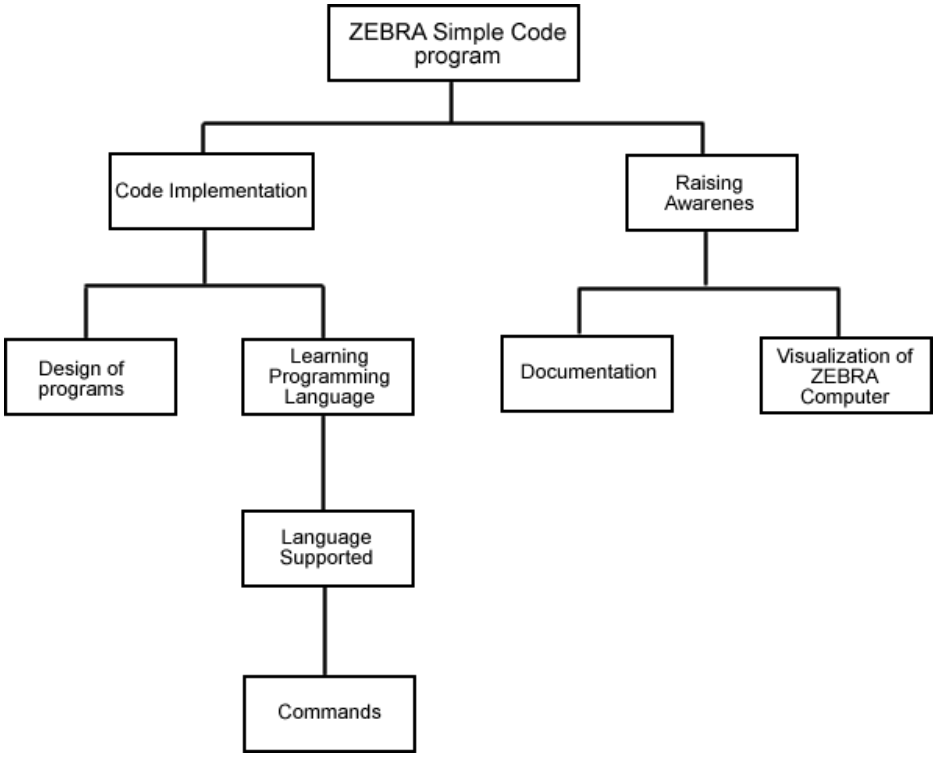


Figure 1.6: *Work breakdown structure for ZEBRA project*

having to revive the computer, its importance and how this will benefit the computing society as a whole.

1.5 Project Scope

The scope of a project defines the boundaries of a project. The information that is going to be delivered and to what extent and what is going to be designed, what it will do and what it will not do is defined as the scope of a project. The scope of the project is on:

1. Early machine architecture
2. Virtual machines
3. Emulators

1.6 Requirements

Defining requirements specifies the capabilities, features or attributes of the project's deliverables. The requirements are given below stating what the emulator will look like, what kind of computations it will perform and which type will be created.

Table 1.1: *Project Requirements*

Requirements	Description
Simple Code program	A Simple Code program must be written using the command and routine functions provided from sample programs that come with the emulator
Must run on the ZEBRA emulator	The written program must be able to run on the emulator to show that the student has understood the programming language
Key words to be used as commands to the program	As illustrated in http://www.memtsi.dsi.uminho.pt/ocr/simple_code_zebra.pdf Simple Code was used to program the machine because it was more simplified by using key words in replace of functions
Interpret sample programs	Student must be able to explain the code that was written for some sample programs written for the ZEBRA emulator

The above requirements may be altered during the course of development to further improve the final project.

SUMMARY

Part I of this thesis introduced the Stantec-ZEBRA computer, its background and the inventor van der Poel a Dutch computer scientist. The description was brief leaving room for more explanations in detail in Part II. The main focus was on specifying the aims and objectives of this project to make a clear demarcation on the path that this project will take. The goal is to try to meet these objectives as this will signify the success or failure of this

project. The plan on how to tackle this project is also given in the form of a WBS. This helps to break down and organise activities. By doing this it will make time management more easy as the student only has to follow the order that he/she has designed in order to finish the project on time. A list of the scope of that the project will be given and finally the requirements are given as the key milestones of the project, what is required by the users of the system.

Figure 1.7: *Willem van der Poel, the creator of the ZEBRA computer. Image source* *https://www.knaw.nl/nl/leden/leden/4658*

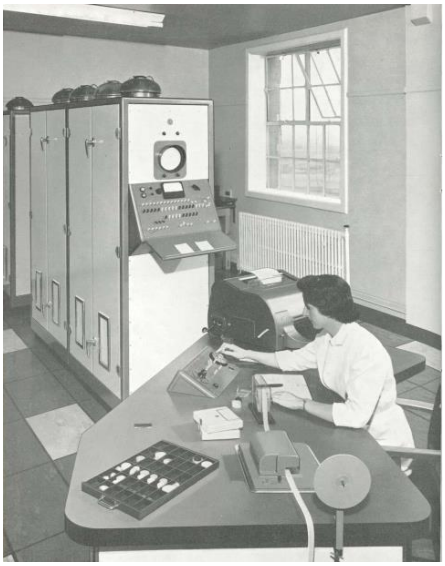


Figure 1.8: *A Stantec-ZEBRA installation in Liverpool, England, used by an animal foodstuffs manufacturer for computation of multiple combinations of vitamin contents against available ingredients at varying prices. Image source* *http://archive.computerhistory.org/resources/text/Standard/Stantec.Zebra.1961.102646082.pdf*

College of Science
Coleg Gwyddoniaeth
Swansea University
Prifysgol Abertawe

Part II

The Stantec-ZEBRA and its Interpretation

- **Chapter 2: UNDERSTANDING THE ZEBRA: AN INTERVIEW WITH ROD DELAMERE**
- **Chapter 3: HOW THE ZEBRA WORKS: AN INTERVIEW WITH ROD DELAMERE**
- **Chapter 4: SIMPLE CODE AND ITS INTERPRETATION**

2 UNDERSTANDING THE ZEBRA: AN INTERVIEW WITH ROD DELAMERE

The main idea of the machine is to economise as far as possible on the number of components by simplifying the logical structure. For example, multiplication and division are not built in but must be programmed (Compiled, Newell, and Bell, 1971).

On the 27th of August 2016 an interview with Mr. Delamere was held by the researcher at his residence in Cardiff. Mr. Delamere is one of the individuals to have used the computer during the 1950s. This interview was carried out in order to get first-hand information on what was and how the machine worked. Mr. Delamere is in possession of a counter book that he used during his University days in the 1950s. In this book he has some Simple Code examples of programs that used to run on the original ZEBRA computer.

Mr. Delamere described the ZEBRA as a simple calculator and reiterated that although they used Simple Code language to write programs that will be put in the machine in the form of paper tape, the language was all not so simple to understand. “The language was called Simple Code, but no, it was not that simple”. According to Mr. Delamere, he once used the ZEBRA machine to calculate the possible trajectory that an aeroplane would use in order for it to land safely without any problems. This calculation would take up to 5 hours on the ZEBRA whereas the time that a plane would take to land in real time was less than 5 min.

A set of equations were internally embedded into the computer and a set on key words, *Y, L0, …,* and *Y,* were used to code the input program. This program was written at the Glamorgan College of Technology in Treforest and the program was transferred onto the paper tape and this paper tape was taken to the Standards Telephone Cables where the computer was and this was input to the machine and the output was given by a tele printer that punched new holes on the paper tape.

The main problem with the ZEBRA was that it had no way of error handling or even giving flags of messages when an error had occurred or what had caused it. The computer’s way of signalling an error was that, during its execution of the paper tape, when it encountered

an error of some sort, be it a misplaced value or a missing value, the computer just halted where it encountered the error. This was the only way that one could tell that there was a problem in executing the code. This was not of much help because an individual would not know what caused the problem and how to even fix it. The only thing that they could do was to check where the tape stopped and trying to find out what values were represented at that point and what could have caused the machine to stop.

This was the main problem when it came to using the computer stated Mr. Delamere. When this happened, you had to go back to the College and rewrite your code paying special attention to the point where the computer stopped. This was a big inconvenience to using the machine and this meant that students had to wait for a week to go back to Standard Telephone Cables to use the computer again.

3 HOW THE ZEBRA WORKS: AN INTERVIEW WITH ROD DELAMERE

The ZEBRA is a calculator that was used to make simple calculations. Although, there exists an emulator for the computer being hosted by the Computer Conservation Society, this emulator seems to be an improvement on the original computer. Mr. Delamere's notebook contains some example programs that he used to run on the original computer together with the results that these equations would output on the tele printer and on the paper tape.

Rod Delamere's Compute 10^x from approximate number $(1 + ax + bx^2 + cx^3 + dx^4 + ex^5)^2$.

Given constants

```
Y          Be ready to accept 'simple code'.
L0         Read e => (0), d => (1), c => (2), b => (3), a => (4).
(Q1) Z     Stop
T100       (A) => (100)      0 => (A)
L6         Read x₁ => 6
+05        Set count to 5 i.e. (α) => δ, 0 => α.
AR         e => (A)    (ex₁ + α)    (ex₁ + d)x + c
V6         ex₁ => (A)  (ex₁ + α)x   etc.
+1         (α) + 1 => α      (β) + (γ) => β    compare (α) with (ε)
A5         ({[(cx₁ + d)x + c]x₁ + b}e₁ + a)x₁ + 1
U10        (A) => (10) (A) remaining
V10        (A)2 => (A)
Z9         C.R.L.F.
Z8         Print contents of (A)
X1         Jump to Q1
Y00        Begin execution at Y
```

The code above computed the results of 10^x using the equation $(1 + ax + bx^2 + cx^3 + dx^4 + ex^5)^2$. This program is known as Simple Code and it produces the computation of 10^x five times. This is achieved by the line +05 which sets count to 5. The parameters for *a, b, c, d* and *e* are (+1.15138424, +0.66130851, +0.26130650, +0.05890681, +0.02936622) respectively and the *x* values are (+0.5, +0.7, 0.25). The output was in the form of tele printer and produced the values (+0. 316665943 + 1, +0.501956281 + 1, +0.1779911374 + 1) together with a paper tape output that is given at the appendix section of this documentation together with other examples.

The computer had a tube that acted as the output window and this valve would only mark memory locations indicating which area contained a value and which one did not, using the (. and !) symbols. The (.) means off or empty no value is present in memory and the (!) means on and a value is present in memory This was the only output available and this did not give away the value that was stored or whether it was a signed or unsigned value. That was one of the difficulties of using the ZEBRA computer. Debugging was made really difficult. Error handling was not even made easy at all.

This computer computed mathematical calculations to precision but it had a main disadvantage which was that it took too long to compute the results. According to Mr. Delamere at one point it had to compute the crystallization of compounds that were used in chemical equations and it took 15 hours to come up with the results of one equation. The other disadvantage of the ZEBRA computer was that it used vacuum tube as those that were used in television sets during that period. These vacuum tubes were fragile and could often explode when the computer was turned on. This meant that the computer had to be left in its on state even for days so as to avoid the damage of these tube because if one was to malfunction it would take about a week to two to replace just one tube and this had an impact on the amount of time that the students will be allowed to use the computer.

The ZEBRA also had a coiled transistor on each and every wire that transferred an electric current over the circuit board. This coiling made it easier to detect where a short circuit will have occurred. Other computers during that period had their transistors glued to the circuit board. This made it really difficult to identify where the short circuit will have occurred.

Apart from that, the time for repairs was reduced by a number of days but still was a lot for students to skip their chances of using the computer.

4 SIMPLE CODE AND ITS INTERPRETATION

Simple Code is a language that was developed to be a really simple form of code for a beginner with further 'not-quite-so-simple' facilities giving a comprehensive and flexible programming code (Ord-Smith, 1960). The only guide that is given to a Simple Code user is the fictitious store and registers which he/she can use. The way that these stores and registers work is similar to how the assembly language works. There are two special registers:

i. An accumulator called A;
ii. An accumulative multiplication register called B.

There are also 6 special registers used for counting and order modification and these are called α, β, γ, δ, ε, θ.

To input numbers in the computer you use the command *Ln* which means, '*read number into n*' and to input a group of numbers you use the instruction *LOn,* which reads the numbers in sequence putting them into *n, n+1, n+2*, … until a terminating *Y* is encountered. The same instruction also counts the numbers, placing the count in δ. The output instructions are *Pn* and *POn* and will be in floating-point form.

4.1 Jump instructions and labels

Simple Code instructions are normally executed in sequence because it is a one-address language (Ord-Smith, 1960). To jump out of sequence of the code, this is done with an instruction *Xp* meaning '*Jump to instruction location labelled p*'. There can be any number of jumps backward or forward to a labelled point. A point may have more than one label attached to it but, the same label cannot be attached to two or more different points. When a jump takes place, the X instruction automatically records a note of the location next to which it occupies and this is called the return instruction and it can be picked up and placed in a specific location by the instruction *XOp*.

There are also two conditional jump or test instructions which are:

- Ep: jump to location labelled p if (A) >= 0 otherwise proceed serially.
- EOp: jump to location labelled p if (A) < 0 otherwise proceed serially.

4.2 Input and execution indications

A Simple Code program when punched on tape is always preceded by the letter *Y*. This is an indication to the computer to jump to the Simple Code interpretive input program and begin taking in Simple Code (Ord-Smith, 1960). There are so many ways to start the Simple Code program and some of them include:

- *Y*: jump to S.C. input, clear labels, and begin input of instructions at the beginning of the instruction store
- *Yp*: means begin input of S.C. without clearing labels at the point previously labelled *p*.
- *YOn*: means begin input (no clearing of labels) of instructions at location *n* in the number store.

Execution indication, *YOO*, means stop input and begin execution at the place given in the last input indication. Therefore, a very simple program will begin with *Y* and be terminated with *YOO*.

To begin a cycle of instructions n times it is necessary to precede the block with +*On* and end it with +1. The operations in the block are then repeated *n* times after which the program proceeds serially. These commands use the special registers α, ε, θ, δ. +*On* does the following:

- The contents of register α are transferred unchanged to register δ: (α) preserved→δ;
- α is cleared;
- The register ε is set to *n*, the count limit
- A return instruction is written into the register θ.

If there are instructions within the loop, the address of which will vary depending on the count, *R* is written before the address in those instructions. An example to sum the contents of locations 200, 201, …, 299, the instructions required are:

```
+O100

AR200

+1
```

The above code has the effect of placing the contents of 200 in *A*, incrementing by 1 (+1) and adding the contents of 201 to *A*, 100 times (+*O*100). The instruction is *A*200. +*n* ('count with *n* at a time') has the action:

$(\alpha) + n \rightarrow \alpha;$

Instructions of the form *IRn* mean the number that is in instruction location *n* + contents of special register α: (*IRn* = *In* + (α)).

4.3 Simple Code in the Real ZEBRA

The interpretive programs of the Simple Code occupy approximately 1000 locations of the real ZEBRA store, a magnetic drum of 8192 words capacity. All the interpretive programs and all the subroutines are completely 'dead', they have no 'live' locations and can be locked away at the end of the store where they will be read but cannot, without unlocking them, be overwritten accidentally (Ord-Smith, 1960). Simple Code instructions are partially interpreted during input and are translated by the input program into a Normal Code instruction pair; one called the address part and one the operation part.

Instructions are classified into two types:

- Extractive instructions and
- Non-extractive instructions

Extractive instructions must extract the number to work with before performing operations, for example *A, S, V, D* and – instructions. Non-extractive instructions have to perform their main operation before worrying about storage location of the number, for example *U, T, L*

and + instructions. Execution of a Simple Code instruction consists of first extracting the appropriate instruction pair and transferring them into the *A* and *B* accumulators of ZEBRA.

A wide variety of problems have been programmed in Simple Code, including telephone traffic problems, cable calculations, aircraft flight calculations and integration, have demonstrated the flexibility of the code and the speed with which such problems could be tackled, and working on calculations involved in what are called uniformity trials for the determination of optimum plot size and shape.

4.4 Stantec-ZEBRA Simple Code Instruction Code

Arithmetic Instructions

- An (A) + (n) → A
- Sn (A) − (n) → A
- Vn (A) * (n) → A
- Nn −(A) * (n) → A
- Dn (A) / (n) → A
- Tn (A) → n 0 → A
- Un (A) → n
- Hn (n) → A

Accumulative multiplication instructions

- Kn (n) → B

Followed immediately by:

- Vn′ (B) * (n′) + (A) → (A)
- Nn′ −(B) * (n′) + (A) → (A)

Also provided:

- V0n (A) + $(n)^2$ → A
- N0n (A) − $(n)^2$ → A

Some of the common subroutines are given below. The full list is given at the Appendix section

- Z stop, wait for dial or start key
- Z7 test key U1

- `Z8` `print (A) in floating form`
- `Z19` `jump from Simple code to Normal code`

These are the most common commands and instructions used in the development of a Simple Code program. All input numbers in the ZEBRA and converted into floating point form for faster processing. This is how the machine is structured (Ord-Smith, 1960).

SUMMARY

Part II of this thesis focuses on the Stantec-ZEBRA computer in more technical detail. Interviews were held with a former user of the ZEBRA computer during the 1950s and he provided first hand source information in the form of his notebook exercises from College and these are presented in Appendix I. This information was the turning point in the development process of this project. This information gave much detail in how to write Simple Code programs. Together with the short paper written by Ord-Smith describing the Simple Code, it made it more easy for the student to grasp the techniques on how the code is written. More advanced sections where presented in van der Poel's book The Simple Code for ZEBRA.

Figure 4.1: *Rod Delamere pictured in May 2008 with his programming exercise book from 1961 for the Stantec-ZEBRA.* Image source http://www.swansea.ac.uk/library/archive-and-research-collections/hocc/peopleandreminiscences/reminiscences/roddelamere/

College of Science
Coleg Gwyddoniaeth
Swansea University
Prifysgol Abertawe

Part III

Software preservation and the Stantec-ZEBRA

- **Chapter 5: LITERATURE REVIEW**
- **Chapter 6: EMULATION AND EMULATORS**

5 LITERATURE REVIEW

5.1 Introduction

This section focuses on looking at similar works that were done throughout the years and to find out a relationship between the work of the student and that which was done before. This is a review of texts from scholars which includes the current knowledge including substantive findings, as well as theoretical and methodological contributions to a particular topic. By referencing to known scholars this will solidify the relevance of the project and to show that what is being done is not totally new but some people have done it or at least attempted to do it and evidence is supplied for it.

5.2 Emulation vs Simulation

In some regards, the words emulation and simulation can be used interchangeably but there is a difference between the two concepts. In order to successfully complete this project, the researcher had to clearly distinguish between the two terms so as to minimise ambiguity and demarcate the bounds and scope of this project.

5.2.1 Emulation

(Joshi, 1989) defined emulation as a means of operating close to real-time. An emulators' goal is for it to replace the original for real use. An emulator uses the same processes to achieve its objective and is mostly made out of the same materials as the original. (Kulenov, 2016) looks at emulation as a model of some system that captures the functional connections between inputs and outputs of the system, based on processes that are the same as, or similar to, those of that system, and that is built of the same materials as that system. (Kumaarr, 2014) is of the opinion that, in computing and electronics, an emulator is considered as a software or hardware which can imitate (duplicate) the behaviour and functionality of different software within another software/hardware platform. Emulation is best described as imitating a certain computer platform or program on another platform or program. In this manner, it is possible to view documents or run programs on a computer

not designed to do so. An emulator is itself a program that creates an extra layer between an existing computer platform “host platform” and the platform to be reproduced “target platform” (What is Emulation, no date).

With emulation, one will be trying to match the exact performance, speed and ability of the original. With this in mind, emulation is slow and expensive to do. One has to properly match the inner behaviours of an original device. If it took time to boot up as a machine, then the emulator also has to take the same amount of time. For an emulator to be considered good enough, it must act as the original device and it must be able to run the same programs as its predecessor at the same speed and using the same amount of resources.

5.2.2 Simulation

Simulation is a way that mimics the activity of something that it is simulating. It appears to be the same as the thing being simulated. For example, the flight simulator "appears" to be a real flight to the user, although it does not transport you from one place to another. A simulator is an environment which models but cannot be substituted with real system (Joshi, 1989). In more technical terms, a simulator is a model of a system that captures the functional connections between inputs and outputs of the system, but without necessarily being based on processes that are the same as, or similar to, those of the system itself (Kulenov, 2016). In a simulator, the operation of a targeted system is recreated to the best possible. The underlying mechanisms used to recreate the scenario may be the same or different from the original. A simulator sets up a similar environment to the original device's OS, but doesn't attempt to simulate the real device's hardware. Some programs may run a little differently, and it may require other changes (like that the program be compiled for the computer's CPU instead of the device's), but it's a close enough match that you can do most of your development against the simulator (Keyan, 2014).

Simulation is mainly focused on trying to reproduce and what is being made can have the same look and feel even the performance but it is not the same as the original device. A good example of how and where simulators work is the flight, car and movie simulators. An individual can enter into a plane or car and inside of it you can find the steering wheels

and controllers and the screens that you look at will give the environment as if you were actually driving or flying and the simulator tends to shake and give an experience of being real but in actual fact it will be stationary.

(Kulenov, 2016) gives a table that gives the distinctions between simulation and emulation in more technical details.

Table 5.1: *The difference between an Emulator and a Simulator*, source The Open University, 2007

Emulation/Emulator	Simulation/Simulator
The microprogram-assisted macroprogram which allows a computer to run programs written for another computer.	One that simulates, especially an apparatus that generates test conditions approximating actual or operational conditions.
Hardware, software or a combination of the two that enables a computer to act like another computer and run applications written for that computer. In the past, it was often a hardware add-on that actually contained an instruction execution module for the emulated computer. Today, "emulator" more often refers to software, which provides a translation layer from the emulated computer to the computer it is running in. The emulator may translate machine language, calls to the operating system or both.	A broad collection of methods used to study and analyse the behaviour and performance of actual or theoretical systems. Simulation studies are performed, not on the real-world system, but on a (usually computer-based) model of the system created for the purpose of studying certain system dynamics and characteristics. The purpose of any model is to enable its users to draw conclusions about the real system by studying and analysing the model. The major reasons for developing a model, as opposed to analysing the real system, include economics, unavailability of a "real" system, and the goal of achieving a deeper understanding of the relationships between the elements of the system.

An emulator in computing duplicates (provides an emulation of) the functions of one system using a different system, so that the second system behaves like (and appears to be) the first system. This focus on exact reproduction of external behaviour is in contrast to some other forms of computer simulation, which can concern an abstract model of the system being simulated.	Simulation can be used in task or situational training areas in order to allow humans to anticipate certain situations and be able to react properly; decision-making environments to test and select alternatives based on some criteria; scientific research contexts to analyse and interpret data; and understanding and behaviour prediction of natural systems, such as in studies of stellar evolution or atmospheric conditions. With simulation a decision maker can try out new designs, layouts, software programs, and systems before committing resources to their acquisition or implementation; test why certain phenomena occur in the operations of the system under consideration; compress and expand time; gain insight about which variables are most important to performance and how these variables interact; identify bottlenecks in material, information, and product flow; better understand how the system really operates (as opposed to how everyone thinks it operates); and compare alternatives and reduce the risks of decisions.
Emulation is the process of mimicking the outwardly observable behaviour to match an existing target. The internal state of the emulation mechanism does not have to	Simulation involves modelling the underlying state of the target. The end result of a good simulation is that the

accurately reflect the internal state of the target which it is emulating. The goal of an emulation is to able to substitute for the object it is emulating.	simulation model will emulate the target which it is simulating. A simulation's focus is more on the modelling of the internal state of the target — and the simulation does not necessarily lead to emulation. In particular, a simulation may run far slower than real time.

The difference between the two, henceforth, is made easy going through the literature of different authors. As mentioned above, an emulator is mainly concerned in reproducing the actual device, behaviour, functionality and speed using materials or software similar to the original and the end result is a functional device, whereas, a simulator mimics the original concept and creates/models an environment same as what the device will encounter but this device cannot be used as a substitute for the real work. A simulators' main purpose is for education and training. Giving you an environment that is the same as to what you expect and training you in advance before doing the actual thing in real life, just like first person shooter games for the army, it's a training exercise to give you more information on how to operate during battle.

This project is, therefore, focused on the Stantec-ZEBRA computer and by deducing the above information it can be found that it matches the criterion of a simulator. This is due to the fact that the software created by Don Hunter, although it is an exact replica, the time that was spent processing on original programs during the 1950s is more than the time that his emulator takes to make mathematical equations. Don Hunter's emulator was designed to run on all operating systems including the Windows 7 and later versions. This is made possible through the use of a software called the DOSBox. This DOSBox simulates the DOS environment on the machine and from this environment the ZEBRA computer can be run without any problems. This DOSBox helps to compile the PASCAL code used to write the ZEBRA with ease and also gives the chance to enthusiastic programmers and scholars

to make changes and help improve the computer or learn the programmatic details of the machine (Computer conservation society, no date).

This Stantec-ZEBRA runs the actual programs that were run on the original ZEBRA machine but with addition of some few lines of code which will be explained in later chapters. These programs were written in Simple Code and Normal Code which is similar to Sub Programming. Sample programs written in these languages are available. This program is software based only and hence any hardware will not be referenced to, however, in the near future other projects can be embarked on as to revive the hardware parts of the machine and have it up and running. An example of such an endeavour is the project to build, demonstrate and maintain a replica of the Small-Scale Experimental Machine (SSEM) - the world's first computer. This project is a currently undergoing activity and there are volunteers needed in helping to recreate the machine and having it up and running at the London Museum. To learn more about this venture, visit these websites http://curation.cs.manchester.ac.uk/digital60/www.digital60.org/rebuild/ and http://www.cs.man.ac.uk/CCS/SSEM/volunteers/index.html.

5.3 Our Digital Heritage

Digital Preservation is the management and maintenance of digital objects which can be categorised into the files, or groups of files, that contain information in digital form, so they can be accessed and used by future users (Stuchell, 2013). This preservation of digital data leads to the upcoming generation's heritage. This heritage is key to people knowing where they are coming from, where they are and where they are heading. There are so many events that are held over the world to celebrate our digital heritage and the recent one is the Digital Heritage International Congress held in Granada Spain from 28 September to 2 October 2015.

These events help to raise awareness and try to convince people to preserve data, whatever in its form because you never know when it will be needed in use and who it will help. Digital data is one of the most vulnerable forms of data due to the fact that if something happens to where it is stored, like a virus attack, the data will be corrupted, altered or completely lost. Recovery from such situations is really hard but there have been

developments of many means of back up as a number of alternatives to CD's, flash and external drives. The Cloud and Google drive are some of the few to mention where you store your digital information and there it is guaranteed safe from corruption and deletion. Apart from these there are organizations who specialise in the conservation of digital information and this section is dedicated in giving more insight on them and their importance as our digital heritage keepers.

Founded in 1989, the Computer Conservation Society is a joint venture between the British Computer Society, the Science Museum and the Museum of Science and Industry in Manchester. This organization's primary mission is to preserve historic computers, develop awareness of the history of computing, and encourage research. It runs many specialised projects, organise public lecture series, and publish a regular bulletin (Computer conservation society, no date).

This organisation is focused on preserving digital data but in the form of application software. Apart from the Computer Conservation Society, there exist some organisations that are committed to preserving digital information. Some of them include:

Table 5.2: *Organisations concerned with preserving digital information*

Digital Preservation (Library of Congress) http://www.digitalpreservation.gov/	The National Digital Information Infrastructure and Preservation Program implements a national strategy to collect, preserve and make available significant digital content, especially information that is created in digital form only, for current and future generations.
University of Kent UK mirror services http://www.mirrorservice.org/	The UK Mirror Service provides a collection of mirrors of FTP, web and rsync sites of interest to academic users. The service is provided by the University of Kent's School of Computing.

Online Historical Encyclopaedia of Programming Languages http://hopl.info/	This site is concerned with the idea-historical treatment of the development of programming languages as a means of human expression and creation.
http://www.cs.man.ac.uk/CCS/	
Software Preservation and Machine Emulation http://sw.ccs.bcs.org/CCs/index.html	In preserving software, the desire is to keep the source text so that the style of programming is visible, and to provide facilities for execution of the preserved software on widely available current platforms. Where source text is in a language unlikely to be widely known (often assembly language) the intention is to provide reference information to enable the source text to be understood by a reader with basic programming literacy.

The (The encyclopaedia of computer languages, no date) states that in 1976, at the History of Computing Conference in Los Alamos, Richard Hamming described why we might be interested in the history of computing: "we would know what they thought when they did it". We need to know why the people who designed programming languages thought the way they did. When they designed languages, they made conscious choices, which we have to live with. And they made those choices for a wide variety of reasons, as many as the reasons for which they felt the need to create a language in the first place.

We see that for every creation there is a reason behind, an influencing factor that drives individuals to do come up with better means and ways of solving particular problems. The invention of programming languages led to the creation of computers and these computers initially were huge, expensive to run in terms of resources and very difficult to use. As more and more individuals studied how these computers worked they found better ways and means of designing them in a new way, to be faster, able to store information electronically, to be compact and portable and eventually to interconnect and for the global village that we

have today. This whole process did not just happen within a day. It took years of studying, improving and reengineering for what we have at this moment to be what it is.

The danger, however, is that technology is evolving at a fast pace and people are now more concerned than ever to keep up with the changing trends and tend to forget the later and it tends to be rendered obsolete. As mentioned by the (The encyclopaedia of computer languages, no date), as the world becomes increasingly and overwhelmingly dependent on software, we find that the core of that software - programming languages and systems - remain a mystery to even their users, the programmers. By paying attention to the origin, rise and fall of each of the languages we may learn why they made their decisions. (Santayana, 1905) also said, "...Progress, far from consisting in change, depends on retentiveness. When change is absolute there remains no being to improve and no direction is set for possible improvement: and when experience is not retained, as among savages, infancy is perpetual. Those who cannot remember the past are condemned to repeat it. In the first stage of life the mind is frivolous and easily distracted, it misses progress by failing in consecutiveness and persistence. This is the condition of children and barbarians, in which instinct has learned nothing from experience....".

As mentioned above, if we cannot remember our past then we are condemned to repeat it. During the time of the creation of the first computer, they had their own view of the world and future than what this generation will be anticipating. By learning how they thought and what they were expecting to achieve you will have a foundation on which to build on that is solidified by ideas of the predecessors. Thus, this is the role of the Digital Preservation Societies and the Computer Conservation Society. These organisations are there to preserve the history of digital information, be it in text form or software form. The aim is to keep safe this information and archive it for future use or upgrade.

The Computer Conservation Society in particular is focused on the preservation of digital information but in its software/application form. The society is dedicated to:

- To promote the conservation of historic computers and to identify existing computers which may need to be archived in the future.
- To develop awareness of the importance of historic computers.
- To develop expertise in the conservation and restoration of historic computers.

- To represent the interests of Computer Conservation Society members with other bodies.
- To promote the study of historic computers, their use and the history of the computer industry.
- To publish information of relevance to these objectives for the information of Computer Conservation Society members and the wider public.

This organisation is not centred on making profits and thus it is funded and supported by voluntary subscriptions from members, a grant from The Chartered Institute for IT (BCS), fees from corporate membership and members' donations. The society is also home to some projects of rebuilding early computers and one of the noted on is the project to build, demonstrate and maintain a replica of the Small-Scale Experimental Machine (SSEM) - the world's first computer. Some other projects being undertaken include the Colossus Rebuild Project, Our Computer Heritage, creating simh tapes images from real tapes, The HEC 1 computer and computer films and other media known to the CCS. For more information on these please visit http://www.computerconservationsociety.org/special.htm.

The Society is also home to the famous computers that have been the pioneers of all the computers that we have in our current day and age. The software is downloadable and sufficient documentation is given explaining how these machines work and what they were used for during their days. The softwares that are housed by the Computer Conservation Society are:

- Manchester Baby (SSEM)
- Cambridge EDSAC
- Ferranti Sirius
- Ferranti Pegasus
- Stantec-ZEBRA
- ICT/ICL 1900
- Elliott 903
- Ferranti Atlas 1
- MU5
- KDF9

- LEO III

One of the many activities of the Computer Conservation Society is to develop and preserve software for historic computers, many of which have long since passed into oblivion. In order to make this activity meaningful and to allow for the possibility of writing new programs for dead computers it is sometimes necessary and always useful to implement emulators: programs which run within modern computers but which interpret programs written for the target machine and cause them to be executed in much the same way as the hardware of the original computers once interpreted the instructions of their programs and carried out those instructions (Computer conservation society, no date).

5.4 Don Hunter's Stantec-ZEBRA emulator

Don Hunter was in charge of the ZEBRA installed at STC's research laboratories (Standard Telecommunication Laboratories Ltd, or STL) in Harlow from 1960 to 1963. He had earlier worked on Edsac I (Society). Hunter was part of the people who installed and first used the Stantec-ZEBRA machine. He wrote an article called (The Stantec-Zebra Computer Memories of the Zebra). In his article he acknowledged the initial works of WL van der Poel. Besides working on the ZEBRA machine van der Poel also worked on Testudo and it took 5 years to complete. It could do in 16 hours what we human beings could do in eight, if we worked intensively for the whole day. The other one is the Ptera (PTT Electronic Reckoning Automat).

The emulator created by Don Hunter mimics the original ZEBRA. Even though the telephone dial, switches, input and output tape and teleprinter are not visible they can still be used on the emulator. The input to them is done via key press on the keyboard. All the major components are accessed via a sequence of key presses and the result is printed on the screen. This emulator is command line based and the help menu can be accessed on the emulator by typing the question mark (?). Output is displayed in the same window of the emulator and can be transferred to the RESULTS.OUT file by the user by following the instructions given in the user manual that comes with the emulator.

Demonstration programs come with the emulator and these help the user to run them and see how the emulator works and its performance efficiency meter which is located at the

bottom right of the screen. These programs are written in Normal Code and Simple Code the programming languages that were used during that time. Other sophisticated programs such as the ACTAB.SRC are provided to show the level of complexity that the emulator can handle and how it can perform.

A more technical and detailed explanation of the emulator can be found online at the Computer Conservation Society http://www.computerconservationsociety.org/resurrection/res11.htm#e the article written by Don hunter.

Figure 5.1: *Alan Marr and Don Hunter at work on the Stantec-ZEBRA installed at STL*: Image source http://www.stlqcc.org.uk/docs/computers_03.htm

6 EMULATION AND EMULATORS

Emulation works by handling the behaviour of the processor and the individual components. You build each individual piece of the system and then connect the pieces much like wires do in hardware. Emulation in itself is broad and can be categorised into software emulation and hardware emulation (How do emulators work and how are they written, 2016). There is strong dependency between hardware and software and this introduces a risk. If one of these fails, it will have influence on the computer's operation and its capabilities. Emulation comes into play by providing a solution to this problem.

As described by (What is Emulation, no date), "emulation is best described as imitating a certain computer platform or program on another platform or program. In this manner, it is possible to view documents or run programs on a computer not designed to do so. An emulator is itself a program that creates an extra layer between an existing computer platform (host platform) and the platform to be reproduced (target platform)." An individual may not be concerned by imitation the actual performance or time measures of hardware or software but so long as the end product acts as required then it is deemed as successful emulation.

Emulation can also be seen as a means of digital preservation and it focuses on recreating the original computer environment. Emulation is difficult, this is because developing an emulator is a precise and time-consuming task and especially because the emulated environment must appear authentic and must function accurately too. The Church-Turing thesis also implies that emulation can be quite difficult, particularly when the exact behaviour of the system to be emulated is not documented and has to be deduced through reverse engineering (Emulator, 2016).

Emulation allows a user to have access to any kind of application or operating system on a current platform, while the software runs as it did in its original environment.

6.1 Benefits of emulation

- Potentially better graphics quality than original hardware.
- Potentially additional features original hardware didn't have.
- Emulators maintain the original look, feel, and behaviour of the digital object, which is just as important as the digital data itself.
- Despite the original cost of developing an emulator, it may prove to be the more cost efficient solution over time.
- Emulators allow software exclusive to one system to be used on another. For example, a PlayStation 2 exclusive video game could be played on a PC using an emulator.

6.2 Shortcomings of emulation

- Due to intellectual property rights, some information is lost in a preservation with little supporting documentation due to the proprietary nature of the hardware and software.
- Copyright laws are not yet in effect to address saving the documentation and specifications of proprietary software and hardware in an emulator module.
- Emulators are often used as a copyright infringement tool, since they allow users to play video games without having to buy the console, and rarely make any attempt to prevent the use of illegal copies and this leads to a number of legal uncertainties regarding emulation, and leads to software being programmed to refuse to work if it can tell the host is an emulator.
- Emulators require better hardware than the original system has.

6.3 Types of Emulators

There are so many types of emulators and it all depends on which angle you look at them. Some emulators are accurate to the core, in terms of clock timing and processing speed and most of them are just developed as a reminder of what used to be and placed in museums for example. Most common types of emulators are game emulators. Game emulators are available for some of these platforms Nintendo, Sony, Microsoft, Sega and Arcade. Popular games like Donkey Kong have been emulated and are being enjoyed by those who used to play the game in its popular days.

Some other emulators come as educational tools such as the Android emulator that comes with Microsoft Android Software Development kit and some come as alternatives and placeholders for the original. Examples include the Blue Stakes and MS Dos.

6.4 What Emulators achieve

Emulators are designed to carry out a specific task and that task is tailor made to the requirements that initiated its creation. This thesis will focus on the historic uses of emulators and these include:

- drive detailed historic analyses to discover and understand designs for computers
- They record inconsiderable detail machines
- They restore an operational understanding of machines and are a tool for investigating questions about the machine and its software
- A tool for historical research for technological developments

SUMMARY

Part III of this thesis talks about the Literature review and highlights on simulators and emulators stating their differences. Focus is also given on the emulator that was developed by Don Hunter which is available on the Computer Conservation Society website. This emulator is a replica of the original ZEBRA computer but it need some extra code to be written to the Simple Code program for it to run a program properly. On this section, a

history of the need for understanding native software and programming languages is given so that readers are well aware of the need of knowing how early programmers thought and plans they had for the future they anticipated.

College of Science
Coleg Gwyddoniaeth
Swansea University
Prifysgol Abertawe

Part IV

Project development

- **Chapter 7: GETTING STARTED**
- **Chapter 8: METHODOLOGY**
- **Chapter 9: TECHNOLOGY CHOICES**
- **Chapter 10: PROJECT PLAN**

7 GETTING STARTED

This thesis will focus on improving/modifying some of the information that was delivered in the first report which gave an overall view of the project. The aim is to write a running program for the Stantec-ZEBRA computer. The objective is to build a program that is able to run the basic programs and implement the functions that the original ZEBRA machine performed. The target is to be able to create a running program and having all of the basic commands functioning properly.

7.1 Refinements/Changes to proposed solution

The initial target for this project was to have as many programs as possible being developed and coded by the student and having to demonstrate them working. This requirement has not changed. In addition, there can be given an explanation for the demo programs created by Don Hunter as a means of showing that the student has understood the Simple Code programming language.

This project is also going to solely focus on Simple Code. Simple Code uses some of the features and procedures of Normal Code (Ord-Smith, 1960). However, normal Code is mentioned in passing and there is no demonstration of its code and functionality. Since the ZEBRA computer is a calculator, mathematical problems will be solved using the Simple code. This will demonstrate how much level of understanding the student has attained.

8 METHODOLOGY

The methodology of choice for the development of the programs is influenced by the already existence of a product (ZEBRA DOS emulator). The product is there and it has gone through the product life cycle phases which are: Development, Introduction, Growth, Maturity and Decline/Death. Not focusing though on the sales of this product, the ZEBRA DOS emulator will be in its maturity phase heading to the decline phase.

There are multiple choices for methodologies that will suit the development of the ZEBRA emulator and these include:

- Waterfall development
- Prototyping
- Spiral development
- V shape model

8.1 Waterfall development

In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. It is also referred to as the linear-sequential life-cycle model. It is mainly centred on verification – are we building the right software, and validation – are we building it right. The phases found in the life cycle include Requirement Gathering and analysis, System Design, Implementation, Integration and Testing, Deployment of system, and Maintenance. You cannot proceed to the next phase before the previous one is complete and at each phase a document has to be produced. Although this model is suitable for small projects it would not be suitable for the program design in Simple Code for the ZEBRA since user involvement is key in its development and requirements are likely to change during development process so going back up a phase to make changes to a specification will be really difficult.

8.2 Prototyping

This model refers to building software application prototypes which display the functionality of the product under development but may not actually hold the exact logic of the original software. With this model multiple designs are made for the interface and as they are presented some of them will be taken off while some of them will be accepted for further improvement. Some of the phases in this model include Basic Requirement Identification, Developing the initial Prototype, Review of the Prototype, and Revise and enhance the Prototype. This model focuses more on user involvement and it goes through the requirements until they are met due to the number of prototypes that will have been developed.

This model is good for the design of the programs that will run on the ZEBRA, however, it is mainly concerned with developing blue or pseudo codes. These pseudo code gives a logical description in detail with all the steps but it will be written in English so that users understand how the logic works. This is not the aim of this project. Pseudo codes are helpful but the English language that will be written will not work if it is used as code for the machine. Rather a second option will be survivable in order to come up with the final running programs for the ZEBRA computer.

8.3 Spiral Development

Spiral model is a combination of iterative development process model and sequential linear development model; waterfall model with very high emphasis on risk analysis. The spiral model has four phases and a software project repeatedly passes through these phases in iterations called Spirals. These phases include Identification, Design, Construct or Build, and Evaluation and Risk Analysis. This model is suited to complex projects and those that have requirements that are ever changing and which are undertaken in a turbulent environment.

This model is fairly suited to all sorts of projects but the major disadvantage for it towards the development of the ZEBRA Simple Code program that management for the project will become very costly and complex. Having the project go through many iterations for risk

and design and identification will take much of the development time. Just by having the specifications clearly labelled out it will be more than enough for the system to be developed successfully.

8.4 V shape model

V model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. The next phase only starts when the previous phase is done. The corresponding testing phase of the development phase is planned in parallel so there are verifications on one side of the V and validations on the other side. Coding phase joins the two sides of the model. The phases found in this model include Business Requirement Analysis, System Design, Architectural Design, Module Design and Coding which is the pivot between verification and validation. The phases above are the verification phases and the validation phases include Unit Testing, Integration Testing, System Testing, and Acceptance Testing.

The advantage of V model is that it is very easy to understand and apply. The simplicity of this model also makes it easier to manage. However, considering the ZEBRA project there is a high risk that the project will not be rigid and that even clearly specified specifications will not remain the same. Once the application is in the testing stage, it is difficult to go back and change a specific function. Flexibility is clearly missing in this model.

Having stated the models and outlined their pros and cons, the ZEBRA development project will be highly suited with the User centred design model/framework. Although the Spiral model will have been a great fit for the project, this project is designed focusing on the needs of particular users and these users included Professor John Tucker and Rod Delamere. These users have an emotional attachment towards it and to use other models to develop the system will be a big risk because these models may not accommodate the personal needs of these users. Also these users have sufficient knowledge of this machine including Mr. Don Hunter and Mr. Noel Cox who are one of the few people to be in physical contact with the machine and used it during its time and also learnt its programming language.

8.5 User Centred Design

User-centered design (UCD) also known as User-Driven Development (UDD) is a framework of processes in which the needs, wants, and limitations of end users of a product, service or process are given extensive attention at each stage of the design process (User-centered design, 2016). The focus on UCD is to get as much detail as possible from the users. First-hand information from them and not to make assumptions and in the end making a product/software that will not confirm to their expectations. According to (Travis, 2015), design is based upon an explicit understanding of users, tasks, and environments. It is driven and refined by user-centred evaluation, and addresses the whole user experience. The process involves users throughout the design and development process and it is iterative in nature.

There are some general phases in a UCD framework and these include:

- Specify the context of use: Identify the people who will use the product, what they will use it for, and under what conditions they will use it.
- Specify requirements: Identify any business requirements or user goals that must be met for the product to be successful.
- Create design solutions: This part of the process may be done in stages, building from a rough concept to a complete design.
- Evaluate designs: Evaluation - ideally through usability testing with actual users - is as, integral as quality testing is to good software development.

There are many variations to the UCD framework and this makes it flexible enough to be incorporated into the other trademark software development models. The one big advantage of using UCD is that it is cheap to use. There are no complicated steps to go through and no need for a fairly large number of people within a team. The goal here is to get enough from the user and that these users participate fully during the development of the software so that what they mention is what they get.

The chief difference from other product design philosophies is that user-centred design tries to optimize the product around how users can, want, or need to use the product, rather than

forcing the users to change their behaviour to accommodate the product (User-centred design, 2016).

9 TECHNOLOGY OF CHOICE

The technology of choice to be used for the development of the Stantec-ZEBRA program is the Simple Code language. Although there can be an option of using assembly language, this will be a very difficult task indeed because there are some built in/hard coded functions in the form of mathematical equations that are not mentioned in Ord-Smith's article and by Don Hunter. This thesis will focus on C and Assembly vs the Simple Code language and give reasons why this language was the number one contender over others.

Table 9.1: *Comparison of C, Assembly and Simple Code in relation to ZEBRA program development*

Topic	**C**	**Assembly**	**Simple Code**
Architecture	It is a structured programming language and knowledge for header files is important to make use of functions	It is a structured programming language and very low level with ability to move values directly in memory	It is a structured programming language; earlier to Assembly and has inbuilt functions to run with Normal Code
Memory management	Manual memory management, when using variables, input data must not exceed the variable size otherwise buffer overflow and overwriting of stake memory will occur	Manual memory management, when using variables, input data must not exceed the variable size otherwise buffer overflow and overwriting of stake memory will occur	Memory management is automatically done by the computer since every entry is converted directly into floating point
Error handling	Minimum error handling as programmer has to write correct code	Minimum error handling as programmer has to write correct code	Very difficult to error handle code; when error occurs the computer just

	otherwise the program will crush	otherwise the program will crush	stops and programmer has to go over the code and find out why it did not run properly
Execution speed	Fast executions time as it is more closely linked to machine code	Very fast execution time as it is linked directly to machine code	Very fast execution time as it is linked directly to machine code and uses only floating point values
Support for other languages	Integration with other languages at the lower level is easy	Integration with other languages at the lower level is easy	Integration with other languages is very difficult because of the missing information on Normal Code
Debugging	Very hard to debug programs since it is structured an error may be flag on a line that will have correct syntax when in actual effect the error occurred on a different line	Very hard to debug programs and must have knowledge of movement of data from one register to another	Very hard if not impossible to debug; full knowledge of the code is needed in order to be able to debug at any point in the program
Interoperability	Programs developed using the C language can be run on any platform	Programs developed using the Assembly language can be run on any platform	Programs developed using Simple Code only run on the Stantec-ZEBRA computer

Support for GUI	C language at programming level has little or no support for GUI. Interaction is command line based	Assembly language at programming level has little or no support for GUI. Interaction is command line based	Simple Code has no support for incorporating GUI

Simple Code was chosen as the technology of choice for this project. Although, Assembly language is fairly similar to Simple Code in terms of moving values directly from one memory location to another, Simple Code has the ability to call Normal Code which is used for mathematical operations using the operator *Z19*. In order to use Assembly language and C language there is a disadvantage in that one has to understand the code that was written to develop the ZEBRA computer. This language is PASCAL.

A line by line interpretation will be needed for the code in order to understand how the computer takes in the Simple Code and switch over to Normal Code for its mathematical operations. The C language also has the problem of memory overflow when a value too big is specified it overflows from one memory location into another and this will cause many errors to occur that are complicated to fix. This is highly possible to happen using the ZEBRA because it only works with floating point numbers and there is a risk that values may be truncated or lose its significance by a certain percentage. So it is safer not to translate the PASCAL code and just use the Simple Code instead.

10 PROJECT PLAN

This phase of the project mainly focuses on how resources are to be allocated towards development and how time will be managed throughout the course of development. The planning aspect is the pinnacle for success for any activity that requires management. According to (Gunder, 2003), planning in itself is the process that identifies the goals or objectives to be achieved, formulates strategies to achieve them, arranges or creates the means required, and implements, directs, and monitors all steps in their proper sequence.

A project is said to be successful when it has met its requirements. These requirements make up the goals of the project and the major goal of this project is to develop a fully functional written program for the Stantec-ZEBRA machine. In order to achieve this, a set of deliverables have to be outlined. These act as the milestones so that progress can be monitored properly. The set of deliverables for the project includes:

- To master software engineering tools and methods and programming in Simple Code language.
- To understand fine details of computer architecture and memories and machine code programming.
- To write a program in Simple Code that can run on Don Hunter's Stantec-ZEBRA emulator.

These are the high level deliverables that are generalised and which touch across all the aspects of the project. The low level deliverables include:

- Learning the Simple Code language in order to write a running program on Don Hunter's emulator.
- Understand sample codes written by Don Hunter as a way of showing that the student fully understands Simple Code.
- Completing the project before the end of the month of September in order to meet the University deadline

Deliverables for them to be effective they need to be given estimated start date and end date and these may be gradually updated during the course of the project. The estimated dates and time frames for the deliverables will be given in the Gantt Chart in figure 10.2. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. This plots the time to be taken versus the deliverables or the task to be accomplished. Below is the Work Breakdown Structure (WBS) that will be taken in order to fulfil project objectives:

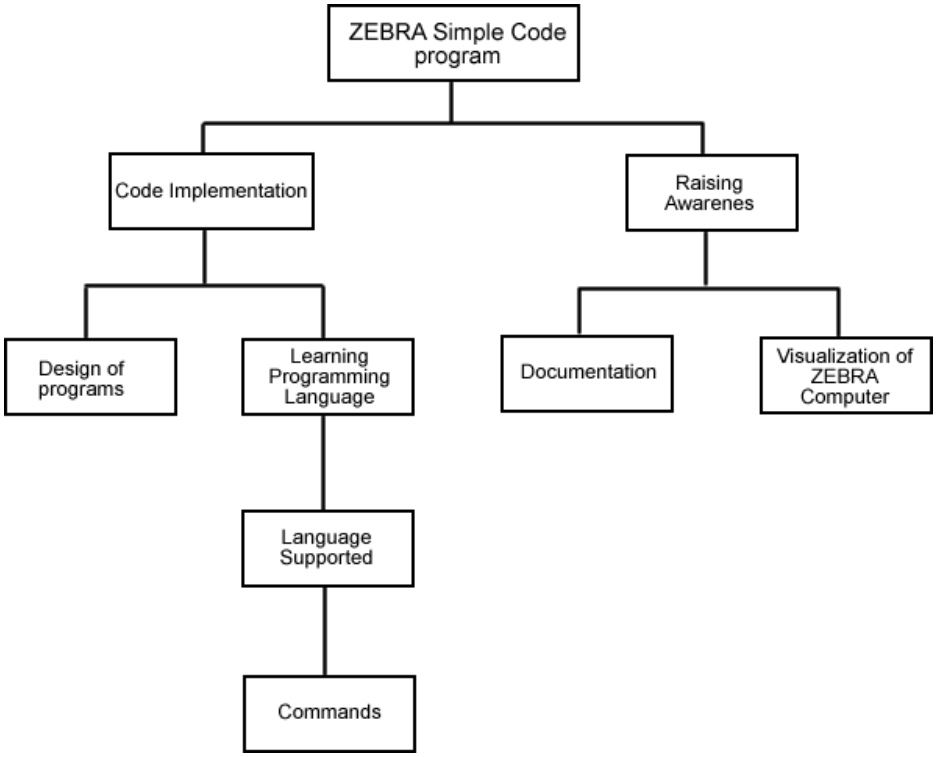


Figure 10.1: *Project WBS*

ID	WBS	Predicessors	Duration Weeks
1	Learning commands	-	3
2	Learning language supported	-	5
3	Learning programming language	1	7
4	Designing the programs	2	2
5	Documentation	1	9
6	Code implementation	2	10
7	Vizualization of the original machine	3	2
8	Testing modules	3	7
9	Raising awareness	6	2

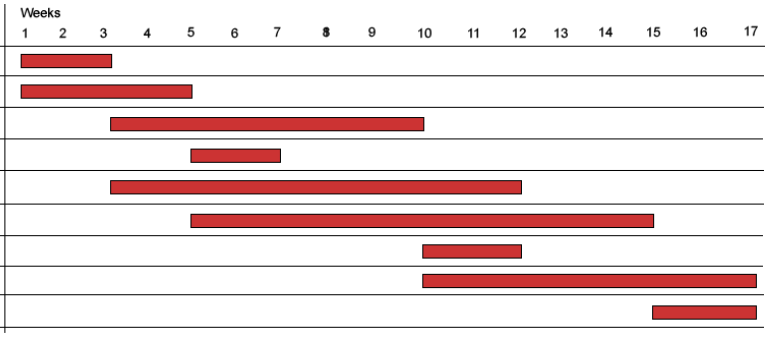


Figure 10.2: *Project Gantt Chart*

The Gantt chart above shows the dependencies and the duration that each and every task is expected to take. The tasks are in the form of WBSs and the estimated durations are listed in weeks required to be spent on each and every task.

The tasks which take the least amount of time which is 2 weeks are Learning commands, Designing the programs, Visualization of the original machine and Raising awareness for the need of such emulators to the public. The tasks that are expected to take up much of the time are writing the documentation and actual coding of the programs which takes 9 and 10 weeks respectively. These are the core activities and thus take about 70% of the duration of the planned time to finish the project. The estimated duration of completing the project is 17 weeks and this starts from June 2016 to mid-September 2016.

The initial tasks are Learning commands and Learning language supported, thus they have no predecessors. After Learning the commands, the next activity is Learning the programming language and this is expected to start whilst still the student is learning the Language supported. At the same time code implementation is expected to start at the same time with Learning the programming language.

Testing of the system is set to start after the Learning programming language is complete and for it to be successful it will be done during code implementation as well.

10.1 Basic Risk Analysis and Management

Risk is defined as the possibility that something bad or unpleasant will happen. Risk cannot be anticipated or predicted when it will occur but there are strategies that will help in formulating plans and proper measures of guarding against the effects of risk. This thesis will use the notion of using the key elements of risk management placing them in a table format, identifying the risk, assessing it, monitoring it and suggesting ways to control or mitigate the effects of the risk.

The scale used for calculating the probability of occurrence of a risk runs from 1 – 5 and 1 meaning that there is a low probability for the risk to happen. The value 3 represent the mid-point that the risk can either happen or not and it will depend if it can be placed under a priority concern or not. The value 5 represents the highest value that the risk may happen and this will require more attention and monitoring during the development of this project.

Table 10.1: *Risk Management*

Risk	Assessment of risk	Monitoring the risk	Controlling the risk	Probability of occurrence (scale 1-5)
Methodology of choice not appropriate	The methodology used for the development of this program may be too complicated and require too much effort to be put in	When duration stated in Gantt chart exceeds for the development of the of the program, this will be the warning of the methodology having failed to meet the needs of the project	Having to take out some of the activities in order to meet project requirements	2

Technology of choice is not meeting requirements	The technology may not be able to implement some of the features and the functions of the original command line emulator	Make sure that functions meet the technology available	Remove features and functions that will be difficult to implement in project development	3
Understanding and implementing Simple Code language	This is a major factor that can lead to the failure of meeting project goals and objectives	Check learning progress against the time stipulated on the Gantt chart to find out if task time has not exceeded original amount of time planned for	Try to put in more effort in order to understand the language because without knowledge of the language then coding cannot occur together with development	3
Time management	Time is a big factor in the success of any project and milestones have to be meet before the deadline because if the	Milestones have to be met before deadline	A pre-allocation of 5 days has been placed on every activity to allow for any complications that may allow a milestone to	4

	overlap into other activities it will cause problems in development		be extended its delivery time	
Illness and personal problems	This will have a small impact in terms of affecting the success of the project unless it is something life threatening	Stick to planned schedules and take regular breaks	Plan to always take a day or 2 off for resting and recuperating in order to maintain the project flow	3

10.2 Testing/Evaluation Plan

The testing of software is an important means of assessing the software to determine its quality. The general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances. The goal of testing is to find an undiscovered error and a successful test is one that uncovers an as yet undiscovered error. Testing is involved in every stage of development but the testing done at each level of software development is different in nature and has different objectives.

Unit Testing is done at the lowest level which is the smallest testable piece of software and it is often called a unit. Integration Testing is performed when two or more tested units are combined into a larger structure. System test is often based on the functional/requirement specification of the system and non-functional quality attributes are also checked. Acceptance Testing is done when the completed system is handed over from the developers to the customers or users.

The testing strategy that is going to be used for the project is black-box testing. This is the technique of testing without having any knowledge of the interior workings of the

application. The tester is looking at the system architecture and will not have access to the source code. While performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

This test strategy is best suited for the project in the fact that what is required of the student is to be able to write a program that will run on the ZEBRA emulator available on the Computer Conservation Society website. Understanding of the code is really crucial but the way in which Normal Code is integrated and how Normal Code is written is not part of the focus for this project.

SUMMARY

The aim of this report is to give a detailed overview of the project. Giving a brief introduction to the problem and picking up the methodology to be used in order to accomplish the task of designing the emulator made up part of the main core of this report. A project evaluation plan was given and this focused on how time and resources will be managed through the use of a Gantt chart and a work break down structure. This project is not intended to be sold and it is meant to be an open source software, therefore, budgeting and finance was not included in this report. The technologies to be used for the development of this project is Simple Code and it was compared and evaluated against other existing technologies to justify its choice as the main technology to be used. Finally, an assessment of the possible risks that might occur was given and strategies were given on how to mitigate then and a scaling factor was given on the probability of the risk occurring and whether it should be priorities or not.

College of Science
Coleg Gwyddoniaeth
Swansea University
Prifysgol Abertawe

Part V

Monitoring, Control and Evaluation

- **Chapter 11: RESULTS AND ANALYSIS**
- **Chapter 12: CONCLUSION**

11 RESULTS AND ANALYSIS

In this chapter the focus is on the outcome of the dissertation as a whole. Emphases will be on whether the requirements and specifications of the project were met. This chapter also gives the author the opportunity to mention the major challenges faced and evaluate the effects of stated challenges and what methods/approaches were taken to overcome these.

11.1 Installing the ZEBRA

The Stantec-ZEBRA emulator developed by Don Hunter is available for download from the Computer Conservation Society. Since technology has changed drastically over time, the ZEBRA was written to be run on MS-DOS or PC-DOS platforms so there is a high chance that by installing the emulator and trying to run it on Windows 7 and earlier versions will prompt the following errors messages (Computer conservation society, no date):

Figure 11.1: *Possible error message 1*: Image source http://www.computerconservationsociety.org/software/dosboxnotes.htm

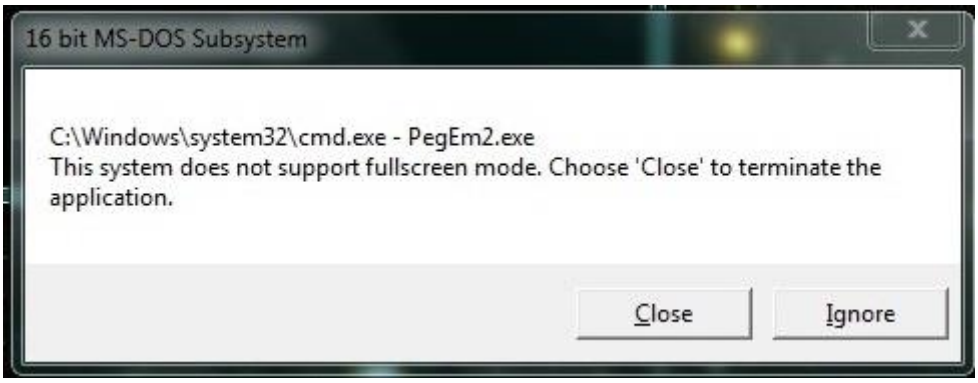


Figure 11.2: *Possible error message 2*: Image source http://www.computerconservationsociety.org/software/dosboxnotes.htm

Figure 11.3: *Possible error message 3*: Image source http://www.computerconservationsociety.org/software/dosboxnotes.htm

The error messages above show that there is a problem that is encountered with running these DOS programs on early computers. It is either that the program is not compatible on 64bit computers or when it is 32bit it will not support full screen mode. On windows 10 it will completely not run and prompt you to find a version of a PC that is compatible to the program.

There is, though, a measure of going around this and it is to download and install DOS-BOX which provides a DOS environment for the DOS programs to run. The DOX-BOS program is available for download at http://www.dosbox.com/. After downloading and installing the DOS-BOX the second step will be, after extracting the contents of the DOX-BOX setup file into a folder, mount the folder created as a virtual hard drive in DOS-Box.

To mount the folder on the hard drive, for example C: drive you write the following in DOS-BOX:

- MOUNT *C C:\[path to the folder with ZEBRA]*
- If the folder name is ZEBRA for example, to run the emulator you just type in ZEBRA and the emulator will run giving the window below
- Type in KEYB UK so that the program will use the UK keyboard layout for input to the program.

Figure 11.4: *Stantec-ZEBRA emulator running*

11.2 Simple Code file format and running a program

All Simple Code programs must be saved using the **.src** extension otherwise it will not run on the ZEBRA emulator. To write a custom Simple Code program the two best programs to do so with are Notepad and Notepad++. To load in a program in the emulator user presses the (a) key and prompts to enter the program name as shown below. An example of a program named **mine4** is entered after the prompt.

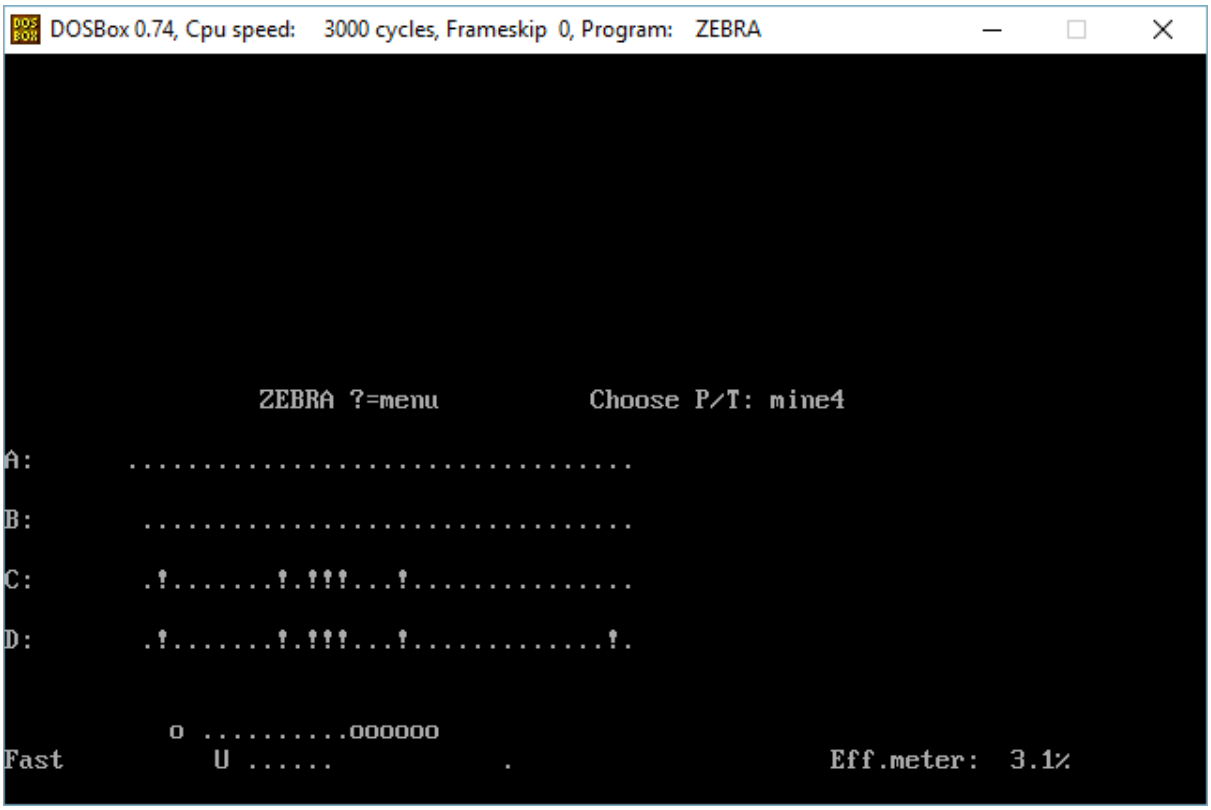


Figure 11.5: *Press a and program askes to input PT paper tape*

To run the program, press *f7* twice and the output from computation will be printed top left to the **ZEBRA ?=menu** as shown below:

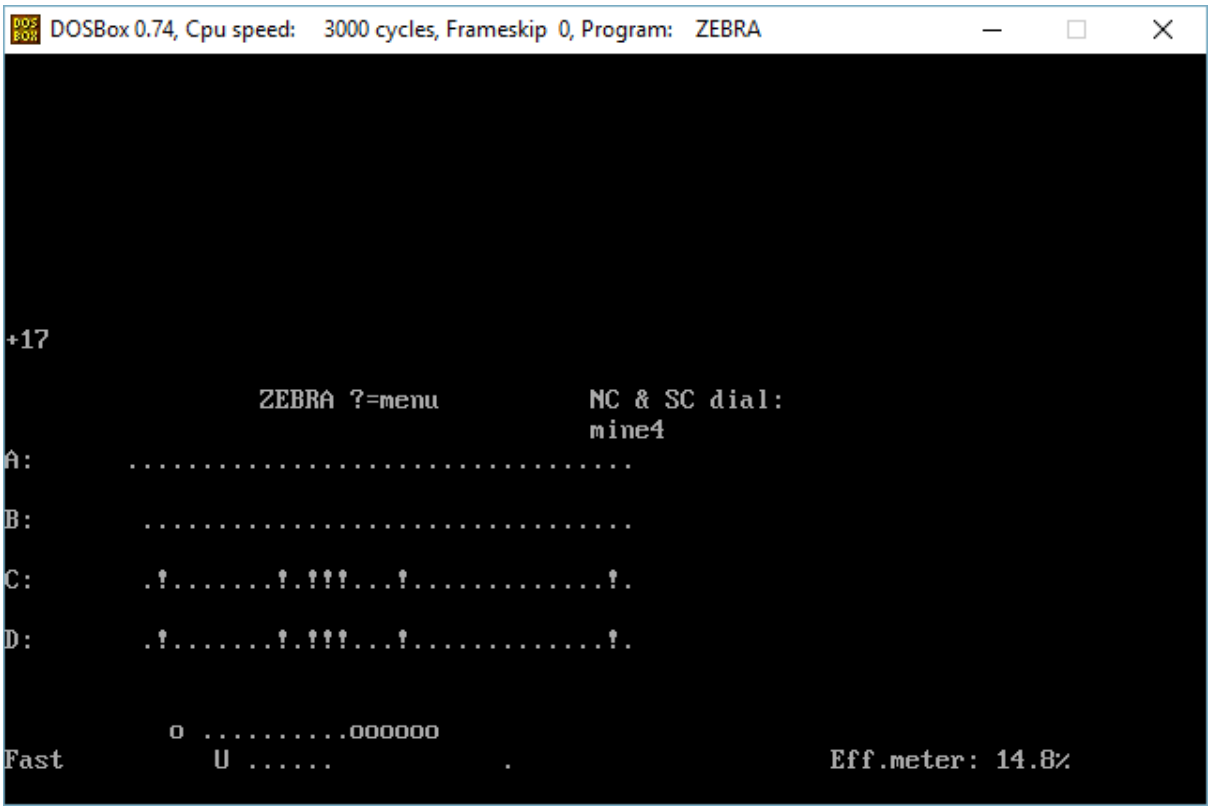


Figure 11.6: *Output of computation on mine4.src Simple Code program*

The dots on the A:, B:, C: and D: lines signify that memory address is empty/off and the question marks means that it has a value/on. NC & SC dial is a prompt to enter a value from the range of 1 – 9 as if you are entering it from a telephone dial. This feature is not used in this dissertation.

11.3 Demo programs

The requirements and specifications of this project were met. The student managed to write his own programs using the Simple Code language. The language is said to be derived from the Edsac I instruction set with the addition of index registers, labels, many built-in functions, a trace facility and nine digit floating point arithmetic and is approximately dated from 1950-55 (Society, no date).

11.3.1 Demo 1

The program takes in a value *c*, multiplies it by the value *a*, and takes the result of (*c***a*) and adds it to b^2. *a* = 2, *b* = 3, *c* = 4, A is the accumulator. The Simple Code for this program is given below:

```
00000
Y
L           {a = +2}
L1          {b = +3}
L2          {c = +4}
-00         {4 decimal digits and no sign}
+00         {no decimal point}
Z30
H2          {move c into A: A contains 4}
V           {A * n = A: n holds the value +2 therefore, A = 8}
V01         {A + (n+1)² = A: (n+1) holds the value +3 therefore, A = 8 +
9 = 17}
Z9          {cr lf fs}
Z31         {print A + (n+1)²}
Z           {stop}
Y00
+2          a
+3          b
+4          c
Y
00?000
```

The first line in any Simple Code program begins with 00000 and this line signifies the beginning of the program. *Y* signifies the beginning of Simple Code. *L*, *L1*, *L2*, mean read number into. *L* has a container *n* which takes an input value. Here *n* = +2, (*n+1*) = +3 and (*n+2*) = +4. Z30 means output or suppress sign. H2 means move value of L2 into A. V means multiply A with *n*. V01 means A plus (*n+1*) and store the contains in A. Z9 prints

out a carriage return. Z31 prints out the value in A with no decimal point. Z stops the computer. Y00 comes straight after Z. Values that are written after Y00 are the values that will be read into n, n+1, etc. Y signifies the end of Simple Code. 00?000 specifies the end of the program.

The output of the above program is +17. A full guide of instructions is given in the appendix section.

11.3.2 Demo 2

Given two sides of a triangle, $a = 3$ and $b = 4$ find the hypotenuse c using the Pythagoras Theorem $c = \sqrt{(a^2 + b^2)}$.

```
00000       {The first seven lines are the same as explained in the above
program}
Y
L
L1
-00
+00
Z30
H
V
V01
Z1          {√ (a² + b²)}
Z9
Z31
Z
Y00
+3
+4
Y
00?000
```

The above program finds the hypotenuse using the Pythagoras Theorem. Given a = 3 and b = 4 therefore, $\sqrt{(3^2 + 4^2)} = 5$.

Note that; in any Simple Code program comment are made inside a program using {}. Anything that is contained between the opening curly brace {and the closing brace} is a comment and is not part of the program. The comments are put inside programs to explain to the reader what the program is or what the line of code means.

11.3.3 Demo 3

Given a triangle in the figure below, find the value of ***a***.

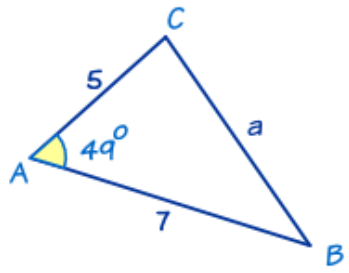


Figure 11.7: *Acute angle triangle, find the value of* ***a***

```
00000
Y
L
L1
L2
L3
L4
L5
H           {n = 49}
Z5          {cos (49) = A}
V1          {A * n+1 = A}
V2          {A * n+2 = A}
V3          {A * n+3 = A}
V04         {A + (n+4)² = A}
```

```
V05          {A * (n+5)^2 = A}
Z1           {√A}
Z9
Z8           {print A in floating point form}
Z
Y00
+49
+7
+5
-2
+7
+5
Y
00?000
```

The value of *a* is supposed to be 5.30 correct to 2 decimal places but the problem is Z5 {cos (49)} is providing an answer of 0.300593 on the ZEBRA computer instead of 0.656059, therefore, the resulting answer from the ZEBRA is wrong.

11.4 Assessment and Evaluation

Looking at the project and comparing the expected results from actual results, it can be concluded that the project was a success. Below are the tables for the requirements that were stated out and the results that transpired during the course of project development. A set of symbols will be used to signify weather the target was met or not using the ✗ to mean that objective was not met and ✓ to show that the objective was met.

Table 11.1: *Requirements against outcomes*

Requirements	Completed	Description
Simple Code program	✓	Provided in this thesis are 3 running examples of mathematical problems.

Must run on the ZEBRA emulator	✓	The three mathematical problems are successfully running on Don Hunter's ZEBRA emulator.
Key words to be used as commands to the program	✓	The key words are the subroutines and procedures that are used in the creation of Simple Code programs and they were used properly in the programs
Interpret sample programs	✓	The student is able to interpret some of Don Hunter's programs but the majority is Normal Code and is difficult to interpret since information on Normal Code is not available

In terms of progress analysis using the WBS and the Gantt chart, all the steps except for one was omitted in the WBS which is the physical visualization of the ZEBRA computer. This did not manage to happen but the student got the opportunity to meet face-to-face with one of the few people to have used the original ZEBRA computers from the 1950s Mr. Rod Delamere. He gave an explanation on what the computer was used for and how the programming language was named Simple Code but yet it was not that simple at all.

With regards to the Gantt chart, changes had to be made in order to finish the project on time. The actual way in which the activities were carried on with looked like this in the end as compared to the planned one under the Project Plan section.

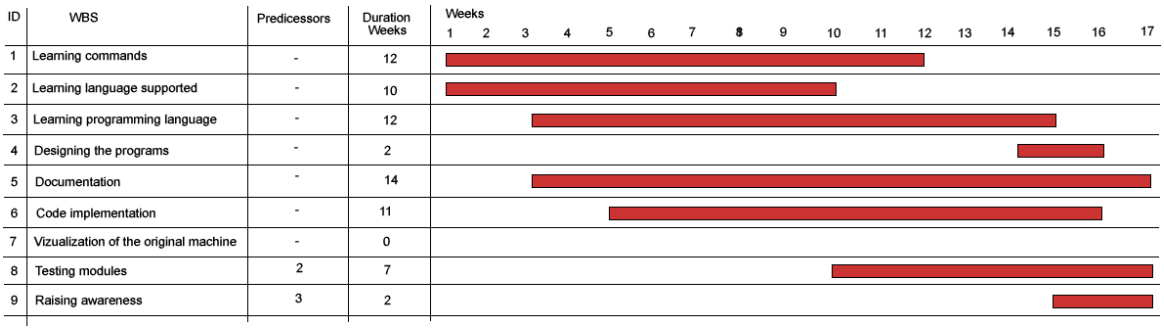


Figure 11.8: *Actual activities and the days taken to complete them shown in revised Gantt Chart*

The writing of the documentation took more than 80 percent of the time and this was the biggest challenge faced by the student. This was followed by learning the commands and the programming language. This had a factor on having the documentation being so long because the student could not start anything before he had a clear mind of the Stantec-ZEBRA computer.

This proved to be a huge challenge because even the Computer Conservation Society was not so familiar on how Don Hunter's simulator worked. Help was eventually acquired from Mr. Rod Delamere but still, his notebook from 1950 together with his examples could not work on the emulator. This took a lot of time for the student to realise that for a program to run on the emulator it requires some additional lines of code.

The major challenge and obstacle faced during the project development was understanding how the Stantec-ZEBRA works. Up until this point, the student has basic functionality understanding of the computer. The fact that debugging the computer is made almost near to impossible without any error detecting systems makes using Simple Code very difficult to even grasp and write a basic program. Below is the revised table for risks that were anticipated and how the student managed to overcome the challenges.

Table 11.2: *Revised Risk Management*

Risk	Assessment of risk	Tackling the risk
Methodology of choice not appropriate	The methodology used for the development of this program may be too complicated and require too much effort to be put in	The methodology of choice (User Centred Design) was an effective choice because there was the right amount of interaction between the users and the student.
Technology of choice is not meeting requirements	The technology may not be able to implement some of the features and the functions of the original command line emulator	Simple Code was the obvious and only choice and it has made the student fully understand how to program in the language

Understanding and implementing Simple Code language	This is a major factor that can lead to the failure of meeting project goals and objectives	Having to understand the Simple Code was not an easy task. It took most of the development time and in the end the student understood the language
Time management	Time is a big factor in the success of any project and milestones have to be meet before the deadline because if the overlap into other activities it will cause problems in development	Time plans had to be changed on a regular basis to accommodate the drastic changes that occurred due to the difficulty in understanding Simple Code
Illness and personal problems	This will have a small impact in terms of affecting the success of the project unless it is something life threatening	Thankfully enough there were no major illnesses witnessed during the development of this dissertation

12 CONCLUSION

During the course of this project development the student has learnt quite a lot from researching, from the supervisor and interacting with the supervisor in order to produce what is required. The highlight of the journey was finally getting to speed with the Simple Code programming language. This took a long time for the student to grasp as it is a very old programming language dating to 1950 and is not even listed on the programming languages list on these popular websites http://rigaux.org/language-study/diagram.html, https://www.levenez.com/lang/ and http://hopl.info/images/genealogies/tester-country.jpg. Simple Code, however, despite the name is not so simple. This code is a set of subroutines that you call using key terms just like how programming works using Assembly language.

Simple Code works directly with numbers in their floating point form which makes it faster for the program to run because there is no need for converting from one form to another which takes up time and memory. It also works by referring directly to a memory location just like Short Code and Assembly code and this makes it really versatile because you are given full control over memory management and have the actual address to every value that is stored in the program.

The biggest challenge that came about using the ZEBRA computer was the problem of debugging. This computer has no means of telling the user the reason of it not working or where the error is or even what sort of error it is. Mr. Delamere concurred to this when he mentioned a story about his encounter with the ZEBRA computer. He said that there was a time when he ran a paper tape program into the ZEBRA and at the middle of the tape the computer just stopped and by that is was a signal that something had gone wrong. At the second attempt he ran the same paper tape and the computer jammed at the same point. At that point in time he did not know what to do next so he took a small piece of paper and stuck it on one of the holes with bubble-gum in the place where the computer was jamming and put back the paper in the computer. At the third attempt, the ZEBRA jammed again but after a nudge of the paper it went through and continued its calculations.

This just shows how hard it was to know what sort of error had effected the computer. It was just too hard to know where to start and the student can testify to this as he also

encountered the same problem when trying to write a simple program on Don Hunter's emulator. It was a gruelling experience because the emulator also gave away no error messages. If the program was wrongly written, for example, the screen would not output anything. Then and only then would you know that something is wrong and the best way to deal with such a case was to go through the program again line by line to see where the problem was.

In the end, three programs were created that are perfectly running on the emulator. There are also many sample programs provided by Don Hunter which show the full power and potential that is held within this computer.

Within the whole journey that the student took on this project, the biggest notable point is that, as technology is evolving we are losing our touch on the obscured or depleted software and programs. By asking online on the explanations on how the ZEBRA works, most of programmers online, especially on Stake Overflow, they did not even know the computer later alone the programming language. This shows that preservation of data is important and raising awareness on existence of such computers had to be done. It might be outdated but by studying it you will get a feel of how early computers worked and it may also inspire you in a way that was not even expected.

As the world becomes increasingly and overwhelmingly dependent on software, we find that the core of that software - programming languages and systems - remain a mystery to even their users, the programmers. By paying attention to the origin, rise and fall of each of the languages we may learn why they made their decisions (The encyclopaedia of computer languages, no date). "...Progress, far from consisting in change, depends on retentiveness. When change is absolute there remains no being to improve and no direction is set for possible improvement: and when experience is not retained, as among savages, infancy is perpetual. Those who cannot remember the past are condemned to repeat it. In the first stage of life the mind is frivolous and easily distracted, it misses progress by failing in consecutiveness and persistence. This is the condition of children and barbarians, in which instinct has learned nothing from experience...." (Santayana, 1905).

12.1 Recommendations

It would be a good thing for the continuation of this project and to design a user interface for the Stantec-ZEBRA computer. Don Hunter's emulator is available for download from the Computer Conservation Society website but it is a DOS software program. Many people are not familiar with using DOS commands, therefore, although the emulator is fully functional it will not be an attractive application to contemporary computer users. An upgrade to a user-friendly interface will enhance its popularity also just like other emulators such as the SSEM and the EDSAC that have user-friendly interfaces that most computer users are accustomed to.

Some work was done by the author regarding the development of a user interface for Don Hunter's Stantec-ZEBRA emulator. Figure 12.1 below shows the work that was done and an individual can pick up form there and finish up the project.

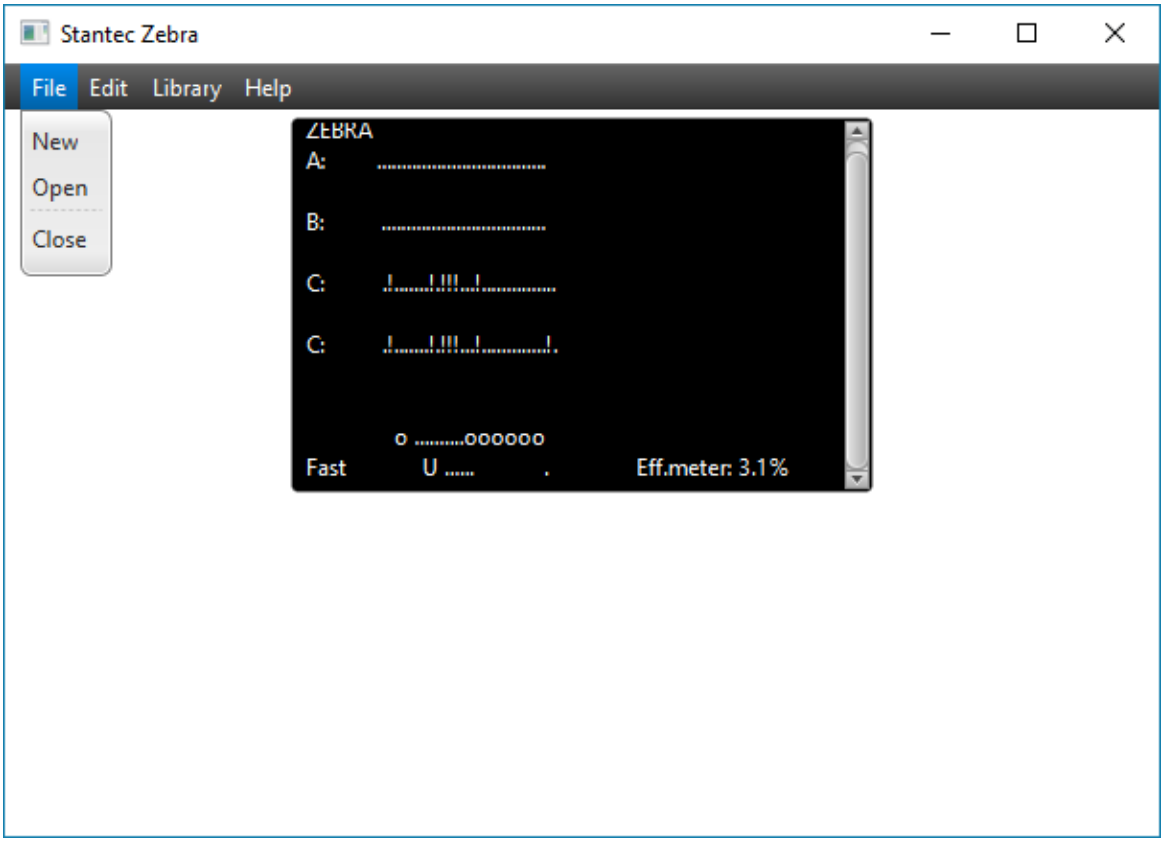


Figure 12.1: *Proposed GUI design for Don Hunter's MS-DOS emulator*

This GUI has been designed using JAVAFX programming language. All of the keys that work on the DOS emulator have been given links in the GUI program however, this program is not functional. It is the skin of the hard code of the ZEBRA emulator and the task will be to interpret Don Hunter's emulator written in PASCAL and try to integrate it with the JAVAFX interface or to redesign the emulator entirely to produce a contemporary ZEBRA computer that can be used in this day and age without experiencing any unnecessary difficulties.

An error handling mechanism must also be put in place so as to give users error and warning messages when they use the machine in the wrong manner. An addition of the telephone dial to the interface will be a great idea so that the full ZEBRA functions will be available to the user.

The second requirement is to embark on a project to reconstruct the ZEBRA in its physical form. To reassemble the original parts and try to build the original computer designed by van der Poel. A similar project is currently underway for the project to build, demonstrate and maintain a replica of the Small-Scale Experimental Machine which is the world's first computer and also the Bombe machine. The SSEM is currently being rebuilt and there are a number of groups who are helping out with this project. To find out more information on these project please visit http://www.cs.man.ac.uk/CCS/SSEM/volunteers/index.html, http://curation.cs.manchester.ac.uk/digital60/www.digital60.org/rebuild/ and http://www.computerconservationsociety.org/wg-bombe.htm.

REFERENCES

1. 2015 (1995) HOPL. Available at: http://hopl.info/ (Accessed: 17 July 2016).
2. 50 years of computing in south wales (2016) Available at: http://www.bcs.org/content/ConWebDoc/37063 (Accessed: 16 August 2016).
3. A little about the STANTEC ZEBRA (1994) Available at: http://niwo.mnsys.org/saved/~flavell/zebra/ (Accessed: 15 September 2016).
4. Afridi, F. (2014) Andy Android emulator free Download. Available at: http://getintopc.com/softwares/emulators/andy-android-emulator-free-download/ (Accessed: 25 July 2016).
5. An Introduction to Stantec ZEBRA (1959) Standard Telephones and Cables Ltd: Information Processing Division, Newport, Mon.
6. BABY DIARY (no date) Available at: http://www.cs.man.ac.uk/CCS/SSEM/volunteers/index.html (Accessed: 30 August 2016).
7. Bauer, Henry R. et al. (1970) Algol W Programmming Manual. University of Newcastle Upon Tyne, Newcastle Upon Tyne, England.
8. Bell, C. Gordon and Alan Newell, (1971) Computer Structures. Readings and Examples. McGraw-Hill Book Company, Inc., New York.
9. Chord (2006) The programming languages Genealogy project. Available at: http://www.everything2.com/index.pl?node_id=858421 (Accessed: 23 June 2016).
10. Compiled, G.B.C., Newell, A. and Bell, G.C. (1971) Computer structures: Readings and examples. 12th edn. New York: McGraw-Hill Inc.,US.
11. Computer conservation society (1900) Available at: http://www.computerconservationsociety.org/software/software-index.htm (Accessed: 08 August 2016).
12. Computer conservation society (no date) Available at: http://www.computerconservationsociety.org/software/dosboxnotes.htm (Accessed: 27 July 2016)
13. Computer conservation society (no date) Available at: http://www.computerconservationsociety.org/software/edsac/base.htm (Accessed: 25 September 2016).
14. Computer conservation society (no date) Available at: http://www.computerconservationsociety.org/special.htm (Accessed: 30 July 2016).
15. Computer conservation society (no date) Available at: http://www.computerconservationsociety.org/wg-bombe.htm (Accessed: 02 September 2016).
16. Computer memories - Stantec zebra Available at: http://www.stlqcc.org.uk/docs/computers_03.htm (Accessed: 03 August 2016).
17. Computing and calculating (no date) Available at: https://www.liverpool.ac.uk/~cmi/src/computers.html (Accessed: 02 September 2016).
18. Diagram & history of programming languages (no date) Available at: http://rigaux.org/language-study/diagram.html (Accessed: 23 June 2016).

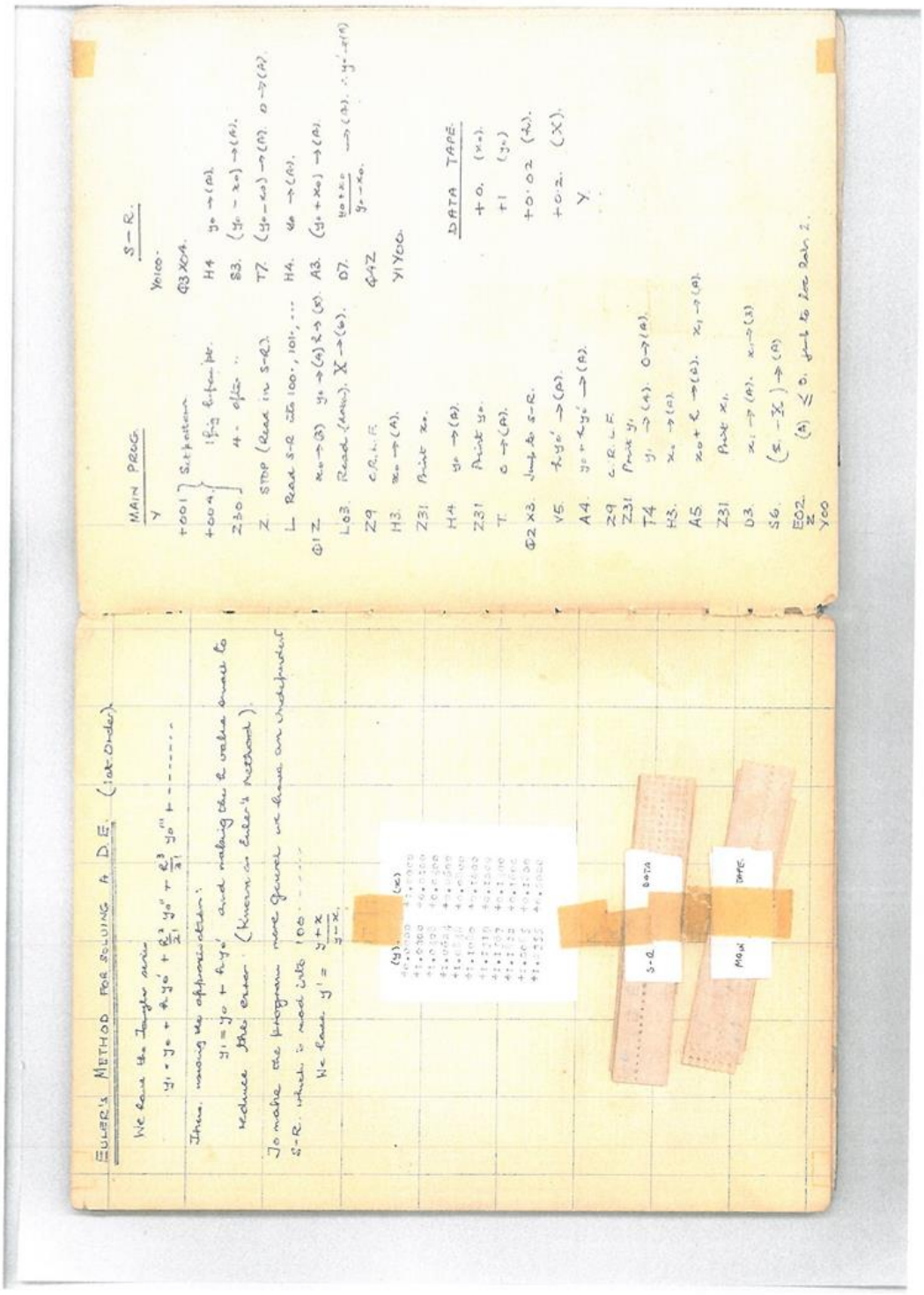
19. Digital preservation (library of congress) (no date) Available at: http://www.digitalpreservation.gov/ (Accessed: 27 July 2016).
20. DOSBox (2016) DOSBox, an x86 emulator with DOS. Available at: http://www.dosbox.com/ (Accessed: 20 September 2016).
21. Download Bluestacks App player for PC - windows 10/8/7 - Download Bluestacks App player (2016) Available at: http://www.download-bluestacks.com/ (Accessed: 25 July 2016).
22. EDSAC (2016) Available at: http://www.tnmoc.org/special-projects/edsac (Accessed: 02 September 2016).
23. Emulator (2016) in Wikipedia. Available at: https://en.wikipedia.org/wiki/Emulator (Accessed: 08 August 2016).
24. Gunder, Michael (2003). Passionate Planning for the Others' Desire: An Agonistic Response to the Dark Side of Planning. Progress in Planning
25. History of Computing Collection at Swansea University (2014) HoCC on BBC radio wales' science cafe. Available at: https://www.youtube.com/watch?v=BrHxCrwR_Ks (Accessed: 16 August 2016).
26. History of Computing Collection at Swansea University (2015) Rod Delamere on the StanTec ZEBRA. Available at: https://www.youtube.com/watch?v=PT2R4GCtAEA (Accessed: 16 August 2016).
27. History of Computing Collection at Swansea University (2016) The Stantec ZEBRA. Available at: https://www.youtube.com/watch?v=GALePEG35VE (Accessed: 16 August 2016).
28. History of programming languages (2016) in Wikipedia. Available at: https://en.wikipedia.org/wiki/History_of_programming_languages (Accessed: 30 August 2016).
29. History of programming languages (no date) Available at: https://web.archive.org/web/20080324234919/http://www.cs.iastate.edu/~leavens/ComS541Fall97/hw-pages/history/ (Accessed: 23 June 2016).
30. HOPL (no date) Available at: http://hopl.info/showhardware.prx?id=4353&which=byhw&Name=Stantec%20Zebra (Accessed: 30 July 2016).
31. Hollingdale, S. H. and G. C. Tootill. (1970) Electronic Computers. Revised Edition. Penguin Books. Ltd., Harmondsworth, Middlesex, England.
32. How do emulators work and how are they written? (2016) Available at: http://stackoverflow.com/questions/448673/how-do-emulators-work-and-how-are-they-written (Accessed: 03 August 2016).
33. Hume, J. N. P. and Beatrice H. Worsley. (1955) Transcode: A system of automatic coding for Ferut. Journal of the Association for Computing Machinery, vol. 2, no. 4, pp. 243 - 252.
34. Hume, J. N. Patterson. (1954) Development of systems software for the Ferut computer at the University of Toronto, 1952 to 1955. Annals of the History of Computing, vol. 16, no. 2, pp. 13 - 19.
35. Joshi, A. (1989) Difference Between Emulation & Simulation. Available at: http://www.slideshare.net/catchanil1989/difference-between-emulation-simulation (Accessed: 25 July 2016).

36. Keyan, K. (2014) What are the differences between simulation and emulation? Available at: https://www.quora.com/What-are-the-differences-between-simulation-and-emulation (Accessed: 25 July 2016).
37. Kulenov, R. (2016) Rustam's techno-feed. Available at: http://blog.avangardo.com/2011/05/what-difference-emulator-simulator-imitator-replication/ (Accessed: 25 July 2016).
38. Kumaarr, H. (2014) What is the difference between emulator vs simulator? Available at: http://hemantcnb.blogspot.co.uk/2013/08/what-is-difference-between-emulator-vs.html (Accessed: 25 July 2016).
39. Lévénez, É. (no date) Computer languages history. Available at: https://www.levenez.com/lang/ (Accessed: 23 June 2016).
40. Marakas, James A. O'Brien, George M. (2010). Management information systems (10th ed.). New York: McGraw-Hill/Irwin.
41. Ord-Smith, R. J. (1960) The STANTEC-ZEBRA and its Interpretation. Annual Review in Automatic Programming, vol. 1, pp. 146 - 168.
42. Ord-Smith, R. J. (1960) The STANTEC-ZEBRA Simple Code and its Interpretation. The Standard Telephones and Cables Ltd.
43. Pardo, C. (2016) Digitisation & digital Archiving. Available at: http://www.dpconline.org/ (Accessed: 17 July 2016).
44. Part17 (no date) Available at: https://webdocs.cs.ualberta.ca/~smillie/ComputerAndMe/Part17.html (Accessed: 03 September 2016).
45. Poel, prof. Dr. Ir. W.L. Van der (Willem) — KNAW (no date) Available at: https://www.knaw.nl/nl/leden/leden/4658 (Accessed: 15 September 2016).
46. Ray, P. (2005). Harrod's librarian glossay and reference book, 10th Edition. Ashgate publisher.
47. Rebuilding the baby (digital 60) (1998) Available at: http://curation.cs.manchester.ac.uk/digital60/www.digital60.org/rebuild/ (Accessed: 02 September 2016).
48. Richard, P (2005). A Glossary of Archival and Records Terminology. Archival Science Society of American Archivists.
49. Rod Delamere (2014) Available at: http://www.swansea.ac.uk/library/archive-and-research-collections/hocc/peopleandreminiscences/reminiscences/roddelamere/ (Accessed: 16 August 2016).
50. Santayana, G. (1905) The Life of Reason. (1 Vols). NEW YORK: DOVER PUBLICATIONS, INC.
51. Sharp, D. (2010) Davidsharp.Com. Available at: http://www.davidsharp.com/baby/ (Accessed: 25 July 2016).
52. Smillie, Keith, (1991). The Department of Computing Science: The First Twenty-Five Years. Technical Report TR 91-01, Department of Computing Science, University of Alberta.
53. Smillie, Keith, (1993). Computing Science at the University of Alberta 1957 - 1993. Department of Computing Science, University of Alberta.
54. Society, C.C. (no date) Computer resurrection issue 11. Available at: http://www.computerconservationsociety.org/resurrection/res11.htm#e (Accessed: 25 July 2016).

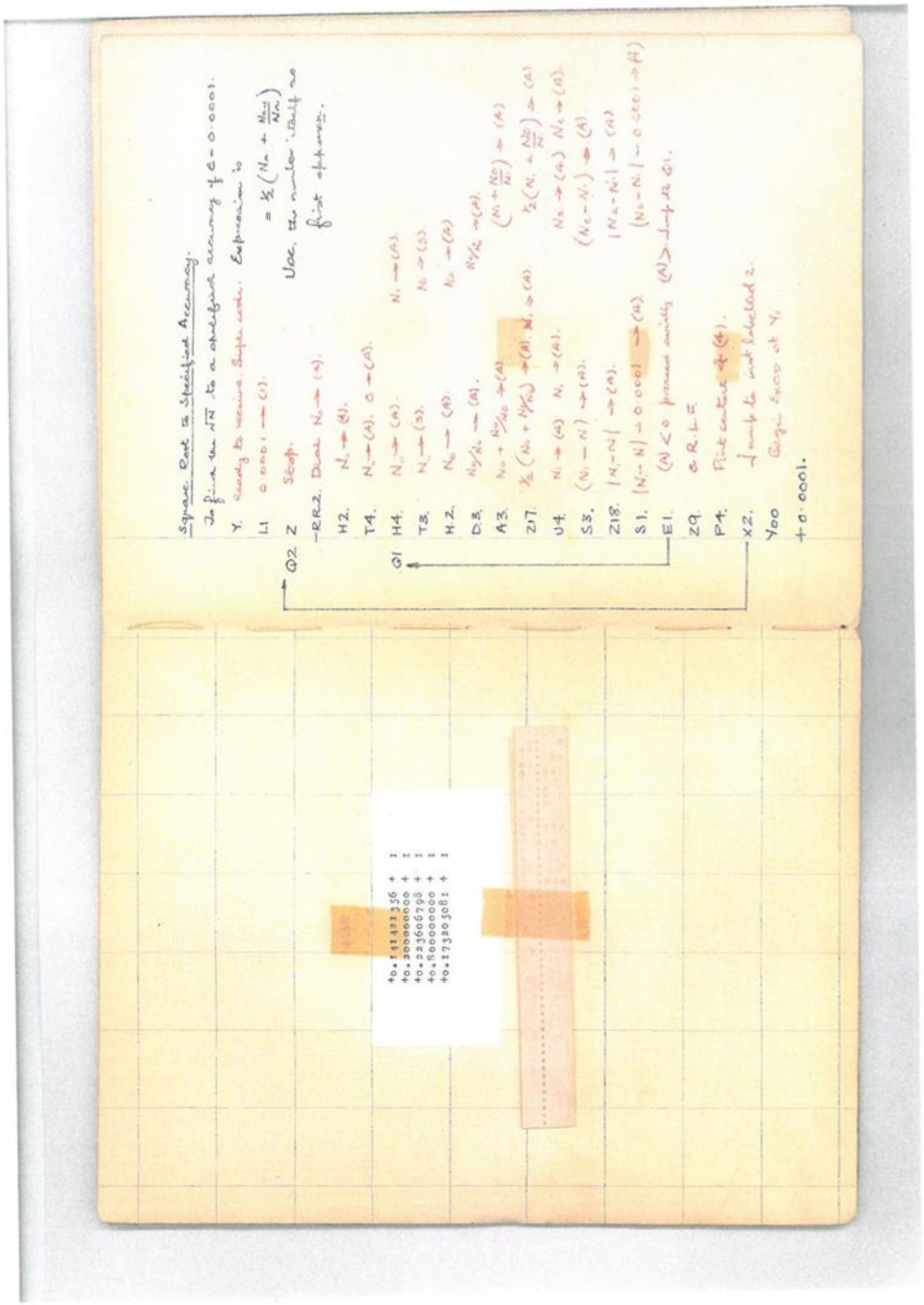
55. Society, C.C. (no date) Computer resurrection issue 16. Available at: http://www.cs.man.ac.uk/CCS/res/res16.htm#d (Accessed: 25 July 2016).
56. Software preservation (1900) Available at: http://sw.ccs.bcs.org/CCs/index.html (Accessed: 30 July 2016).
57. Sommerville, I. and Sawyer, P. (1997). Requirements Engineering: A Good Practice Guide. Chichester: John Wiley & Sons.
58. Standard Telephone and Cables (1961) STANTEC ZEBRA Electronic Digital Computer. Available at: http://bitsavers.trailing-edge.com/pdf/stantec/Standard.StantecZebra.1957.102646083.pdf (Accessed: 30 July 2016).
59. Stantec ZEBRA (2014) Available at: http://www.swansea.ac.uk/library/archive-and-research-collections/hocc/computersandsoftware/earlycomputers/stanteczebra/ (Accessed: 16 August 2016).
60. Stuchell, L.T. (2013) What is digital preservation? Available at: http://www.lib.umich.edu/preservation-and-conservation/digital-preservation/what-digital-preservation (Accessed: 27 July 2016).
61. The computer conservation society (UK) (no date) Available at: http://www.cs.man.ac.uk/CCS/ (Accessed: 21 September 2016).
62. The encyclopaedia of computer languages (no date) Available at: http://hopl.info/why.html (Accessed: 17 July 2016).
63. The UK mirror service (no date) Available at: http://www.mirrorservice.org/ (Accessed: 27 July 2016).
64. The virtual museum of computing (VMoC) (1950) Available at: http://www.si.mahidol.ac.th/simi/museum.html (Accessed: 15 September 2016).
65. Travis, D. (2015) User centred design. Available at: http://www.userfocus.co.uk/consultancy/ucd.html (Accessed: 08 August 2016).
66. User, S. (2016) Introduction - digital preservation coalition. Available at: http://handbook.dpconline.org/introduction (Accessed: 17 July 2016).
67. User-centred design (2016) in Wikipedia. Available at: https://en.wikipedia.org/wiki/User-centered_design (Accessed: 08 August 2016).
68. van de Mey, G. and Laboratorium, N. (1962) Process For an Algol Translator Part One: The Translator. Available at: http://bitsavers.trailing-edge.com/pdf/stantec/Zebra_Algol-60_Part1_Jul62.pdf (Accessed: 30 July 2016).
69. van de Mey, G. and Laboratorium, N. (1962) Process For an Algol Translator Part Two: The Interpreter. Available at: http://bitsavers.trailing-edge.com/pdf/stantec/Zebra_Algol-60_Part2_Jul62.pdf (Accessed: 30 July 2016).
70. van de Mey, G. and Laboratorium, N. (1962) Process For an Algol Translator Part Zero: Introduction Part Three: The Tables. Available at: http://bitsavers.trailing-edge.com/pdf/stantec/Zebra_Algol-60_Part0_Part3_Jul62.pdf (Accessed: 30 July 2016).
71. van der Poel, W. L. (1956) The Logical Principles of Some Simple Computers. University of Amsterdam
72. van der Poel, W. L. (1959) THE SIMPLE CODE FOR ZEBRA.
73. Virine L, Trumper M. ProjectThink: Why Good Managers Make Poor Project Choices. Gower Pub Co.

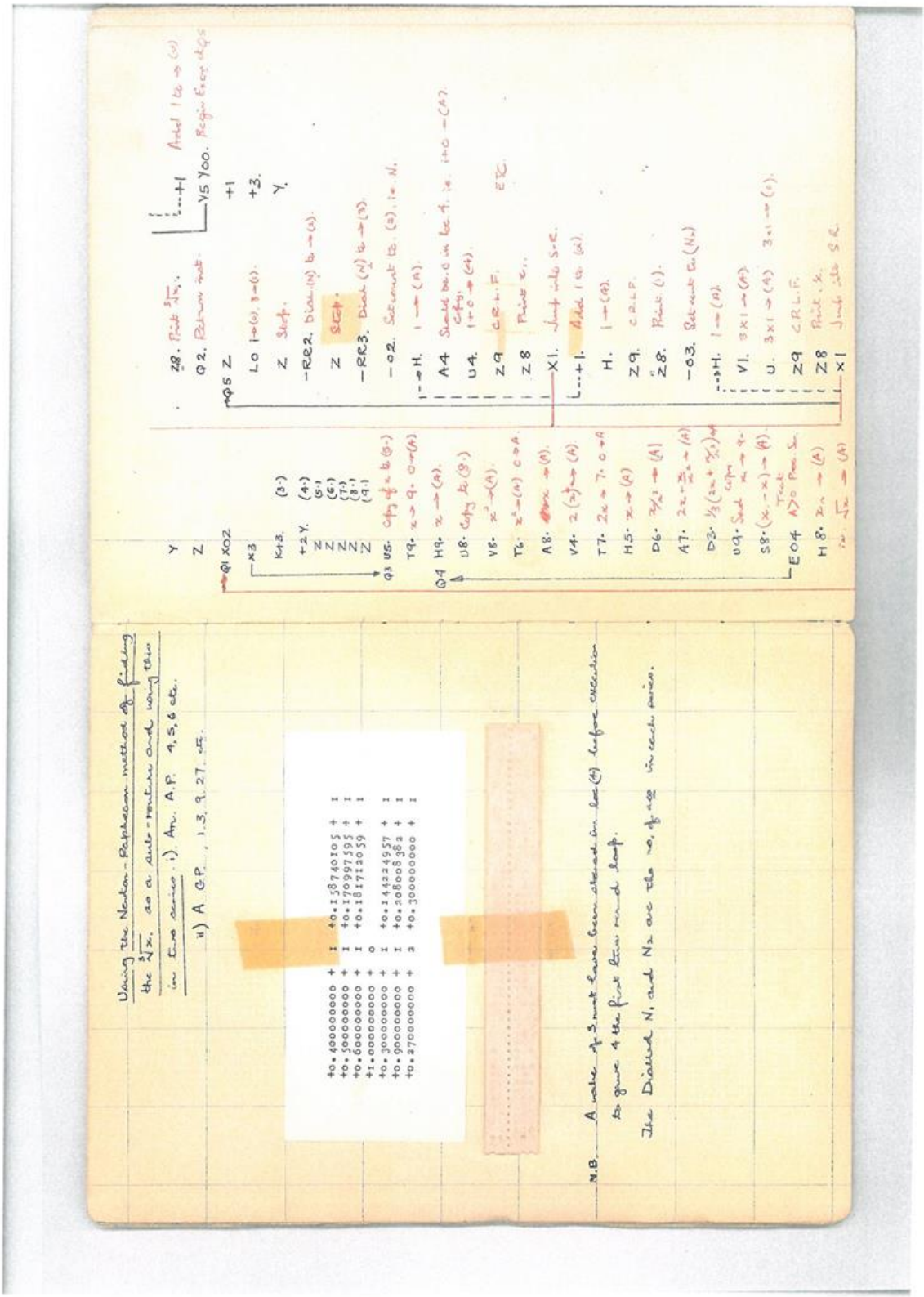
74. Weaver, Patrick (2006). "A Brief History of Scheduling." Mosaic Project Services Pty Ltd.
75. What is Emulation (no date) Available at: https://www.kb.nl/en/organisation/research-expertise/research-on-digitisation-and-digital-preservation/emulation/what-is-emulation (Accessed: 25 July 2016).
76. Wilson, James M. (2003). "Gantt charts: A centenary appreciation". European Journal of Operational Research.
77. Ziring, N. (1997) Dictionary of programming languages. Available at: http://cgibin.erols.com/ziring/cgi-bin/cep/cep.pl (Accessed: 23 June 2016).

APPENDIX I: ROD DELAMERE'S SIMPLE CODE EXECISES

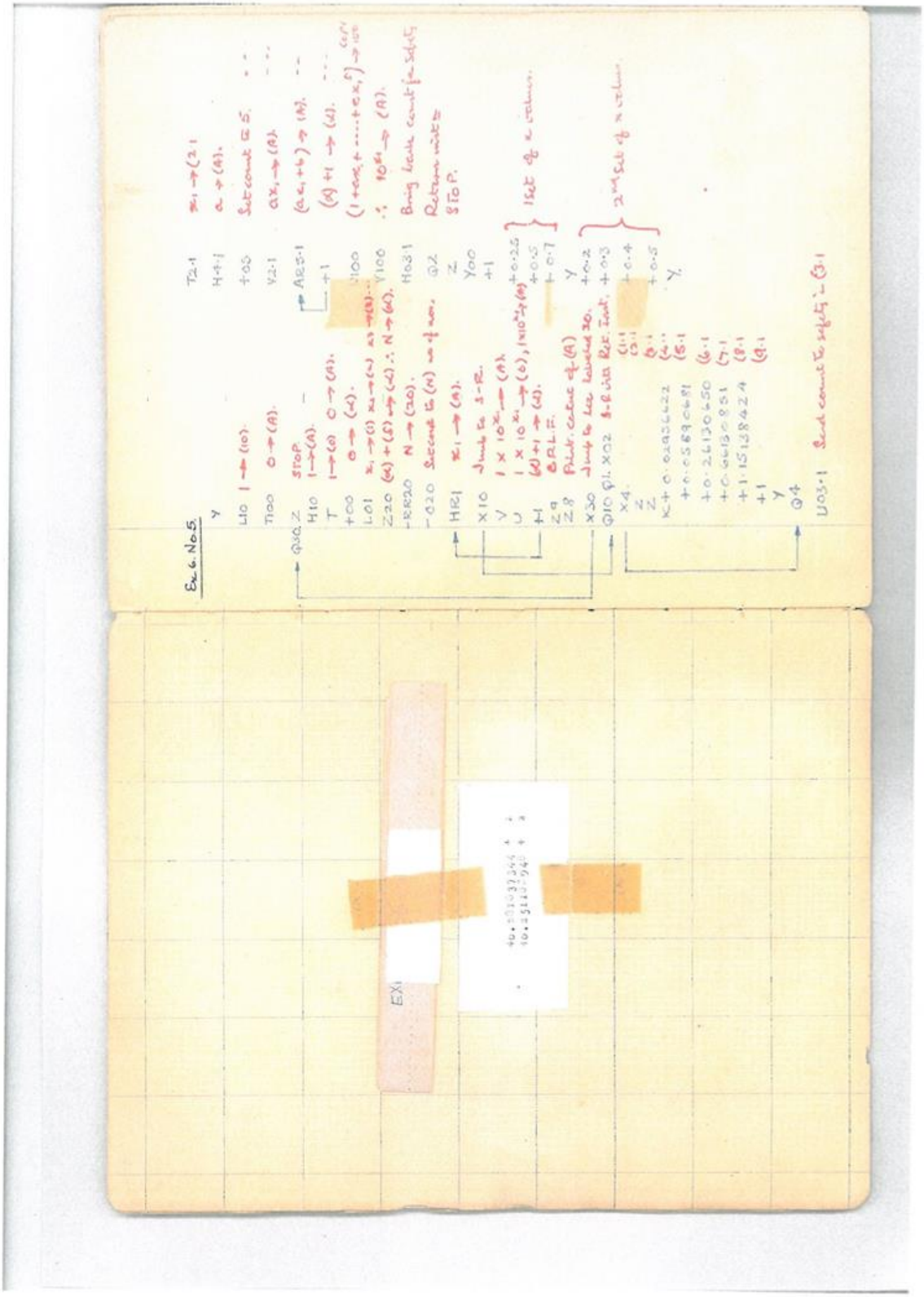


Linear Interpolation.
Given a set of values x1, x2, ... xN
(N ≤ 50) and each output y1, y2, ... yN
and X values Xn ≤ X ≤ Xn+1
Write a program to calculate y by the linear interpolation between
(xn, yn) and (xn+1, yn+1).
Print out one line in floating pt form at each example y value.
Formula for y is:
y = yn + (X - Xn)/h (yn+1 - yn)
using absolute test.
where, h = Xn+1 - Xn.
+0.320000000 + 0
+0.320000000 + 0
+0.340000000 + 0
+0.220000000 + 0
+0.124645200 + 1
+0.140532800 + 1
+0.112783400 + 1
Main Program
Data a)
Data b)
Y: Distance from S Coords
Z
L01 Read in (x1, y1) ... (xN, yN)
L0100 yn, (xn, yn), (xn+1, yn+1) etc.
Z Set Index Type b)
L10 Read X, → 10
H10. X1 → (A)
S1 (X1 - Xn) → (A)
E01. Test -ve > 0
H10 Xn → (A).
S2. (X - Xn) → (A).
E1. Test -ve < 0
H10. Xn → (A).
S1. (X - Xn) → (A)
D3. X - Xn / h → (A)
U15. X1 - Xn / h → (15) also Y in (A)
-0015 Set X to Index [Q]
-RR20. [Q] → 20. Index → Set Pt.
H15. Y → (A).
S20. Q - [Q] → A. Jobe. a).
E 2. Test Y (A) < Previous
H 20 [Q] → (A)
S 4. [Q]-1 → (4).
T20. [N]-1 → 20 → +1
Q2, H 20. (Q)-1 → (A) Q → (A).
V3. Q, P. → (A)
A1 Q, P. + X1 → A Jobe b).
∴ Xn → (A)
T25. Xn → 25 → (A)
-0020 Set X to Q.
HR100 [Index+1] → (A) yn → (A)
T30 yn → Bal → (A)
HR101 [Index (A)] → (A) yn+1 → (A)
S30. (yn+1 - yn) → (A)
T35. (yn+1 - yn) → (35) → (A).
H10. X → (A).
S25. X - Xn → (A)
D3. X - Xn / h → (A)
V35. X - Xn / h (yn+1 - yn) → (A)
A30. yn + X - Xn / h (yn+1 - yn) → (A)
Z9. C.R.L.F.
P10. Print x value.
Z8 Print corresponding y value
X1 Return for another X.
Y00. Accept answer as Y.
Jobe. a) out.
1·10517
1·16182
1·22140
1·28403
1·34986
1·41907
1·49182
1·56831
1·64872.
Y
X = 0·22, 0·34, 0·55, 0·12.


Square Root to Specified Accuracy.
Y. Ready to receive Subroutine.
L1 0.0001 → (1).
Z Stop.
-R.R2. Data N0 → (2).
H2. N1 → (4).
T4. N2 → (A). 0 → (A).
H4. N3 → (A).
T3. N4 → (3).
H2. N5 → (A).
D3. N6/N1 → (A).
A3. N7 + N8/N0 → (A).
Z17. ½(N0 + N9/N0) → (A). N1 → (A).
U4. N10 → (4). N2 → (A).
S3. (N3 − N1) → (A).
Z18. |N4-N| → (A).
S1. (N5 − N) − 0.0001 → (A).
E1. (A) < 0 branch exactly (A) > 0 to R1.
Z9. to R.L.=.
P4. Print contents of (4).
X2. Jump to next Subroutine 2.
Y00 Begin Loop at Y.
+0.0001.
Q2
Q1
+0.141423356 + I
+0.20000000 + I
+0.223606798 + I
+0.200000000 + I
+0.173205081 + I


Using the Newton-Raphson method of finding the √x, do a desk-routine each using this in the series. i) Ans. A.P. 4, 5, 6 etc.
ii) A.G.P., 1.3, 9, 27, etc.
+0.400000000 + 1 +0.138740105 + 1
+0.500000000 + 1 +0.170997595 + 1
+0.600000000 + 1 +0.181712059 + 1
+1.000000000 + 0
+0.300000000 + 1 +0.142249457 + 1
+0.900000000 + 1 +0.208008382 + 1
+0.270000000 + 2 +0.300000000 + 1
N.B. A value of 3 must have been stored in loc (4) before execution to give 4 the first time round loop.
The Divisor N, and N2 are the +0, +00 increments.
Z8. Print Str.
Q2. Return instr.
Y5 Y00. Regen Error stops
Y
Z
Q1 X02
Q3 U5.
Q4 H9.
E04 A>0 Proc Sr.

To write a program to compute the cube root of any number using
the Newton-Raphson scheme. Expression is:
Y.
= ½[2xn+1 + xn+1/xn²]
L01 Read 2 → (3), 3 → (2).
Z Stop.
-RP3. dialled integer to 3. i.e. x → (3).
H3. x0 → (A).
T5. x0 → (5) 0 → (A).
H5. x0 → (A). ; x1 → (A).
T4. x0 → (A). ; x1 → (A).
H3. x0 → (A). ; x → (A).
D4 x/x0 → (A). ; x/x0 → (A)
D4 x/x0 → (A). x/x0 → (A)
T6 x/x0 → (6) 0 → (A); x1/x0² → (6) 0 → (A).
H4 x1 → (A). x1 → (A).
V1 2x0 → (A). 2x1 → (A).
A6 2x0 + x/x0² → (A). 2x1 + x1/x0² → (A).
D2. ½(2x0 + x/x0²) → (A). x1 → (A); ½(2x1 + x1/x0²) → (A). x1 → (A).
U5. x1 → (5) x1 → (A); x2 → (5) x2 → (A).
S4 (x1 → x2) → (A). (x2 − x1) → (A).
E01 (A) > 0 proceed serially. (A) ≤ 0 jump to 001.
Z9 CB. LF.
P5. Print √x0 ; +2
P3. Print x0 ; +3.
X2 Jump to 02. ; Y.
Y00
+1.000000000 + 0
+0.139921057 + 1
+0.144224957 + 1
+0.138740105 + 1
+0.399999999 + 1
+0.200000000 + 1
+1.000000000 + 0
+0.200000000 + 1
+0.300000000 + 1
+0.400000000 + 1
+0.640000000 + 1
+0.800000000 + 1



ZEBRA PROGRAMMING EXERCISES.

Ex. 6. No 4. Compute 10^{th} from theorem $(1+ax+bx^2+cx^3+dx^4+ex^5)^{-2}$

Given constants

```
Y   Be ready to receive 'single code'
L0  Read a→(0), d→(1), c→(2), b→(3), a→(4).
Z   Start.
T100  (A)→(100)  0→(A).
L6  Read x→(A).
+05.  Set count to 5; i.e. begin (a)→δ, 0→α.
→AR  ε→(A) (ex+d)  (ex+d)x+c      5→ε.
V6  εx→(A)  (εx+d)x.  etc.        Repeat 1→0.
+1  (α)+1→α.  (β)+(γ)→β.  compare (α) with (ε)
A5  ([[(εx+d)x+c]x+b]x+a)x+1   (α)≠(ε) return.
U10  (A)→(10)  (A) working.       (α)=(ε) proceed naturally.
V10  (A)²→(A).
Z9  c.R.L.F.
Z8  Print contents of (A).
→X1.  Jump to Q1.
Y00  Begin execute at Y.
+0·2236622  e)  +0·5.
+0·0589681  d)  +0·7   } x values.
+0·2613060.  c)  +0·25.
+0·6613085 1  b)  +.
+1·1513842 4.  a)
+1
Y
```

+0.316665943 + I
+0.501958281 + I
+0.177991374 + I

APPENDIX II: STANTEC-ZEBRA SIMPLE CODE INSTRACTION CODE

STANTEC-ZEBRA SIMPLE CODE INSTRUCTION CODE

Arithmetic instructions

		Time in milliseconds	
An	$(A) + (n) \rightarrow A$	40	average
Sn	$(A) - (n) \rightarrow A$	40	
Vn	$(A) \times (n) \rightarrow A$	35	
Nn	$-(A) \times (n) \rightarrow A$	35	
Dn	$(A) \div (n) \rightarrow A$	55	
Tn	$(A) \rightarrow n \quad 0 \rightarrow A$	20	
Un	$(A) \rightarrow n$	20	
Hn	$(n) \rightarrow A$	20	

Accumulative multiplication uses:

Kn	$(n) \rightarrow B$	20

followed immediately by:

Vn'	$(B) \times (n') + (A) \rightarrow A$	55

or by

Nn'	$-(B) \times (n') + (A) \rightarrow A$	55

Also provided:

VOn	$(A) + (n)^2 \rightarrow A$	55
NOn	$(A) - (n)^2 \rightarrow A$	55

Subroutines (here numbers in brackets are approximate times in millisecs.)

Z	stop, wait for dial or start key		Z21 $-(A) \rightarrow A$	(15)
			Z22 Punch (A) in floating form	
Z1	$\sqrt{(A)} \rightarrow A$	(80)	Z23 Punch *cr lf fs*	

12

Z2 exp $(A) \rightarrow A$ (variable)
Z3 ln $(A) \rightarrow A$ (150)
Z4 sin $(A) \rightarrow A$ (140)
Z5 cos $(A) \rightarrow A$ (140)
Z6 arctan $(A) \rightarrow A$ (variable)
Z7 Test Key *U*1 (15)
Z8 Print (A) in floating form
Z9 'Print' *cr lf fs*
Z10 log $(A) \rightarrow A$ (180)
Z11 arccos $(A) \rightarrow A$ (500)
Z12 sinh $(A) \rightarrow A$ (360)
Z13 cosh $(A) \rightarrow A$ (360)
Z14 arccosh $(A) \rightarrow A$ (430)
Z15 arctanh $(A) \rightarrow A$ (400)
Z16 2 $(A) \rightarrow A$ (30)
Z17 $\frac{1}{2}$ $(A) \rightarrow A$ (20)
Z18 $|(A)| \rightarrow A$ (15)
Z19 Jump from *S.C.* to *N.C.*
Z20 Special counting facility (15)
Z24 Fixed-point number in floating form
Z25 Floating-point number in fixed form
Z26 Punch or print space (which and the number depends on (α))
Z27 Test key *U*2 (15)
Z28 $1/(A) \rightarrow A$
Z29 Punch (A) in fixed-point form according to pattern
Z30 Set fixed-point pattern. (Can output or suppress sign and describe number of digits before and after point)
Z31 Print (A) in fixed-point form according to pattern.

Z32–Z63 can also be used and will contain subroutines peculiar to the needs of a particular user, i.e. these call-in instructions can be changed to meet particular requirements.

Input/output instructions

		Time in milliseconds
Ln	Read number $\rightarrow n$	10 msec/digit + 40 msec/number
LOn	Read numbers $\rightarrow n, n+1, \ldots$ terminated with a *Y*	10 msec/digit + 25 msec/number
Pn	Print (n)	
POn	Punch (n)	

See also Z8, Z9, Z22, Z23, Z26, Z29, Z30, Z31 in list of subroutines.

Control instructions

		Time in milliseconds
Xp	jump to location labelled *p*, remember return instruction in special place	15
XRRROp	jump to location labelled *p*, remember return instruction in register θ	10

XOp	place return instruction set by last *Xp* in location labelled *p*	20
Qp	label location *p*	
QOp	clear label *p*	
Ep	jump to location labelled *p* if $(A) \geqslant 0$, otherwise proceed serially	15
EOp	jump to location labelled *p* if $(A) < 0$, otherwise proceed serially	15

Input

Y clear labels (if *Y* is at the beginning of tape) and begin input at the beginning of the instruction store
Yp begin input at location labelled *p*
YOn begin input of instructions in the number store at location *n*
YOn· begin input of instructions in the instruction store at location *n*.

Execution indication

YOO begin execution at the place given by last input indication

Pointed address facilities

In· number is in instruction location *n*.
In·p number is in that instruction location which is the *n*th relative to location labelled *p*
(*I* is any arithmetic, input or output instruction)

Relative address facilities

IRn interpreted as $In + (\alpha)$
IRRn interpreted as $In + (\beta)$
IRRRn interpreted as $In + (\alpha) + (\beta)$
R can be attached to *A, S, V, N, D, T, U, H, K, VO, NO, L, LO, P, PO, X, XO, E, EO*
RR can be attached to all the above *except XO, E, EO*
RRR can be attached similarly with the further exclusion of *X*

Counting facilities

For details see text.

$+On$ set count to n times
$-On$ set count to $[(n) + \frac{1}{2}]$ times
$+n$ count with n at a time
$-n$ count with $((n) + \frac{1}{2})$ at a time
UOn put count to safety in *n*

	Also interchange (α) and (β)
HOn	bring back count from safety
	Also $(\alpha) \rightarrow \beta$
$+OOn$	$(\alpha) \rightarrow \delta\ n \rightarrow \alpha$
$-OOn$	$(\alpha) \rightarrow \delta\ [(n) + \frac{1}{2}] \rightarrow \alpha$
$+OOOn$	$n \rightarrow \gamma$
$-OOOn$	$[(n) + \frac{1}{2}] \rightarrow \gamma$
*Z*20	Increase running count and count limit

Additional facilities

DOn	exponent (A) + exponent $(n) \rightarrow$ exponent (A)
DOOOOOn	exponent (A) − exponent $(n) \rightarrow$ exponent (A)
$-RRn$	(α) floating $\rightarrow n$
K followed by numbers *Y*	input of numbers with instructions
QZ	stop during input and begin again with start key
QOOn	jump to drum location *n* during input. This allows change from *S.C.* input to *N.C.* execution to take place immediately
UOOO	jump to drum location = (α) enable jump from *S.C.* to
QOOOn	jump to drum location *n* *N.C.*
*X*43*P*	*Normal Code* instruction allows jump back to *S.C.* after exit with *UOOO* or *QOOOn*.
*X*38*P*	*Normal Code* instruction allows jump from *N.C.* to *S.C.*
*Z*19	*Simple Code* subroutine allows jump back to *N.C.* after exit with *X*38*P*
I	written after *Simple Code* instruction gives 'cut-open' facility (see text)

College of Science
Coleg Gwyddoniaeth
Swansea University
Prifysgol Abertawe

APPENDIX III

RECORD OF SUPERVISION

NB: This sheet must be brought to each supervision and submitted with the completed Dissertation

(to be completed as appropriate by student and supervisor at the end of each supervision session, and initialed by both as being an accurate record. NB it is the student's responsibility to arrange supervision sessions and he/she should bear in mind that staff will not be available at certain times in the summer)

Student Name: CHENGETAI KADENGE

Student Number: 846900

Dissertation Title: UNDERSTANDING THE STANTEC-ZEBRA

Supervisor: PROFESSOR JOHN TUCKER

Supervision	*Date, duration*	*Notes*	*Initials Supervisor*	*Initials student*
1: Brief outline of research question and preliminary title (**by pre June**)	17 June 1hour	Presentation of research outline & Preliminary title	JT	CK
2: Discussion of detailed plan and bibliography **(by June)**	30 June 1hour	Project Plan presented	JT	CK
3: Progress report, discussion of draft chapter **(by August)**	19 Aug 1hour	Proposed draft presented	JT	CK
4: (optional) progress report **(by September)**	16 Sep. 1 hour	Fourth draft presented	JT	CK
5: Submission **(by 30 September)**	30 Sep		JT	CK

College of Science
Coleg Gwyddoniaeth
Swansea University
Prifysgol Abertawe

Statement of originality

I certify that this dissertation is my own work and that where the work of others has been used in support of arguments or discussion, full and appropriate acknowledgement has been made. I am aware of and understand the University's regulations on plagiarism and unfair practice.

Signed: ..... Date: 29/09/2016